

Quick answers to common problems

Web Development with Django Cookbook

Over 70 practical recipes to create multilingual, responsive, and scalable websites with Django

Aidas Bendoraitis

[PACKT] open source
PUBLISHING

目錄

介绍	0
目录.md	1
第一章 Django1.6入门.md	2
第二章 数据库结构.md	3
第三章 表单与视图.md	4
第四章 模板与JavaScript.md	5
第五章 定制模板过滤器与标签.md	6
第六章 模型管理.md	7
第七章 Django CMS.md	8
第八章 层次化结构.md	9
第九章 数据的导入与导出.md	10
第十章 附加功能.md	11

简介

译文同时以 GitHub Issue 的形式发布，点此阅读。

版权协议

如果你认为译文对你所有帮助，而且希望看到更多，可以考虑小额捐助。

Web.Development.with.Django.Cookbook

中文名：《Django网站开发Cookbook》，14年10月份的新书，可不是09年的那个 Django book
原版英文：https://www.packtpub.com/web-development/web-development-django-cookbook
作者：Aidas Bendoraitis
日期： October 2014
特色：Over 70 practical recipes to create multilingual, responsive, and scalable websites with Django
级别：Cookbook
页数：Paperback 294 pages

该书的第二版会在**2016**年二月份上市，具体情况看**Packpub**出版社的连接：**https://www.packtpub.com/web-development/web-development-django-cookbook-second-edition**，既然都出第二版了还是等新版出来再更新吧

第五章预览

第五章 定制模板过滤器和标签

本章，我们会学习以下内容：

- 遵循模板过滤器和标签的约定
- 创建一个模板过滤器显示已经过去的天数
- 创建一个模板过滤器提取第一个媒体对象
- 创建一个模板过滤器使URL可读
- 创建一个模板标签在模板中载入一个QuerySet
- 创建一个模板标签为模板解析内容
- 创建一个模板标签修改request查询参数

简介

众所周知，Django有一个非常庞大的模板系统，拥有模板继承，改变具体值的过滤器，以及属于显示逻辑的标签。此外，Django允许你在应用中添加你自己的模板过滤器和标签。定制过滤器或者标签应该位于应用中 `templatetags` 下的标签库文件。你的模板库在任何模板中使用 `{% load %}` 模板标签载入。在这一章，我们会创建多个实用的过滤器和对模板编辑带来更多控制的标签。

遵循模板过滤器和标签的规定

如果你么有坚持按指南来做，那么定义模板过滤器和标签会变得极其混乱不堪。模板过滤器和标签应该尽可能地服务于模板。它们应该同时具有方便性和灵活性。在这个做法中，我们会看到用来增强Django模板系统功能的规定。

如何做

扩展Django模板系统时遵循以下规定：

```
1. 当在视图，上下文处理器，或者模型方法中，页面更适合逻辑时，不要创建或者使用定制模板过滤器、标签。你的页
```

```
2. 使用 _tags后缀命名模板标签库。当你的app命名不同于模板标签库时，你可以避免模糊的包导入问题。
```

3. 例如，通过使用如下注释，在最新创建的库中，从标签中分离过滤器：

```
# -*- coding: UTF-8 -*-
from django import template
register = template.Library()
### FILTERS ###
# .. your filters go here ..

### TAGS ###
# .. your tags go here..
```

4. 通过纳入以下格式，创建模板标签也可以被轻松记住：

for [app_name.model_name]：使用该格式以使用指定的模型

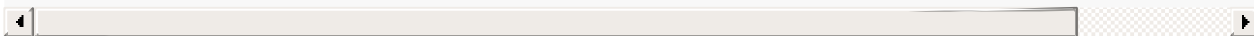
using [template_name]：使用该格式将一个模板的模板标签输出

limit [count]：使用该格式将结果限制为一个指定的数量

as [context_variable]：使用该结构可以将结构保存到一个在之后多次使用的上下文变量

5. 要尽量避免在模板标签中按位置地定义多个值，除非它们都是不解自明的。否则，这会有可能使开发者迷惑。

6. 尽可能的使用更多的可理解的参数。没有引号的字符串应该当作需要解析的上下文变量或者可以提醒你关于模板标签组



参见

创建一个显示已经过去天数的模板过滤器

创建一个提取第一个媒体对象的模板过滤器

创建一个人类可理解的*URL*的模板过滤器

创建一个接纳已经存在的模板的模板标签

创建一个载入模板中的*Queryset*的模板标签

创建一个可以把内容解析为模板的模板标签

创建一个修改*request*查询参数的模板标签

创建一个显示已经过去天数的模板过滤器

不是所有的人都需要持续追踪日期，当论及前沿信息的创建和修改时，对于我们多数人来说，读取时间的差异会更加便利，例如，三天之前发布的博客文章，当日出版的新闻头条，昨日最后登录的用户。此做法中，我们会创建一个称为 `days_since` 的模板过滤器，它会转换

到人类可理解的时间差。

准备开始

如果完成操作，可以在设置中的 `INSTALLED_APPS` 下创建 `utils` 应用。然后，在这个应用中创建命名为 `templatetags` Python包（Python包是一个拥有空的 `__init__.py` 文件）。

怎么做

用下面的内容创建一个 `utility_tags.py` 文件：

```
#utils/templatetags/utility_tags.py
# -*- coding: UTF-8 -*-
from datetime import datetime

from django import template
from django.utils.translation import ugettext_lazy as _
from django.utils.timezone import now as tz_now
register = template.Library()

### FILTERS ###

@register.filter
def days_since(value):
    """ 返回天和值之间的天数."""

    today = tz_now().date()
    if isinstance(value, datetime.datetime):
        value = value.date()
    diff = today - value
    if diff.days > 1:
        return _("%s days ago") % diff.days
    elif diff.days == 1:
        return _("yesterday")
    elif diff.days == 0:
        return _("today")
    else:
        # Date is in the future; return formatted date.
        return value.strftime("%B %d, %Y")
```

工作原理

如果像下边这样在模板中使用这个过滤器，它会返回 `yesterday` 或者 `5 days ago` 这样的内容:

```
“{% load utility_tags %}
{{ object.created|days_since }}
You can apply this filter to the values of the date and datetime types.
```

你可以应用这个过滤器到 `date` 的值，和 `datetime` 的类型。

每个标签库都有一个注册器，它是收集过滤器和标签的地方。Django过滤器使用 `register.filter` 装饰器所注册的函数。默认，模板系统中的过滤器命名为相同名称的函数，或者其他的可调用对象。如果你想的话，你可以通过传递 `name` 到装饰器来为过滤器设置不同的名称:

```
@register.filter(name="humanized_days_since")
def days_since(value):
    ...
```

过滤器本身意思是非常显而易见的。首先，当前日期被读取。如果给出的过滤器值是 `datetime` 类型，那么日期就会被提取。然后，计算今天和提取值之间的差异。视天数而定，返回不同的字符串结果。

还有更多

过滤器可以很容易地扩展以显示时间的不同，比如 `just now` ， `7 minutes ago` ，或者 `3 hours ago` 。只要操作 `datetime` 值便是，而不是操作 `date` 值。

参阅

创建一个提取第一个媒体对象的模板过滤器

创建一个人类可理解的*URL*的模板过滤器

创建一个提取第一个媒体对象的模板过滤器

想象一下，你正在开发一个博客概述页面，对于每篇文章，你想在这个页面中显示内容中的图片，音乐或者视频。这样的情况下，你需要从文章的HTML内容中提取 `<img>` ， `<object>` ，和 `<embed>` 标签。这个做法中，我们会见到在 `get_first_media` 中如何使用正则表达式完成目标。

准备开始吧

我们从位于设置中的 `INSTALLED_APPS` 的 `utils` 应用开始，并将 `templatetags` 包置于该应用内部。

如何做

在 `utility_tags.py` 文件中，添加以下内容：

```
#utils/templatetags/utility_tags.py
# -*- coding: UTF-8 -*-
import re
from django import template
from django.utils.safestring import mark_safe
register = template.Library()

### FILTERS ###

media_file_regex = re.compile(r"<object .+?</object>|"
    r"<(img|embed) [^>]+>") )

@register.filter
def get_first_media(content):
    """ 返回html内容中第一个图片或者flash文件 """
    m = media_file_regex.search(content)
    media_tag = ""
    if m:
        media_tag = m.group()
    return mark_safe(media_tag)
```

工作原理

当数据中的HTML内容有效时，把下面的代码写到模板里，它会从对象的内容字段重新取回 <object> ，<img> 或者 <embed> 标签，或者一个因媒体不存在而出现的空字符串:

```
{% load utility_tags %}
{{ object.content|get_first_media }}
```

首先，我们把编译过的正则表达式定义为 media_file_regex ，然后，在过滤器中，我们执行正则表达式模式的搜索。默认，结果会把出现的 < ， > 和 & 转义为条目 < ， > ，以及 & 。我们使用 mark_safe 函数把结果标记为安全的HTML，为在模板中不实用转义显示做好准备。

还有更多

你可以非常容易地扩展这个过滤器，它也可以提取 <iframe> 标签（最近它们被Vimeo和Youtube用来内嵌视频）或者HTML5的 <audio> 和 <video> 标签，可以像这样修改正则表达式:

```
media_file_regex = re.compile(r"<iframe .+?</iframe>|"
    r"<audio .+?</ audio>|<video .+?</video>|"
    r"<object .+?</object>|<(img|embed) [^>]+>") ″
```

参阅

创建一个显示已经过去天数的模板过滤器

创建一个人类可理解的URL的模板过滤器

目录预览

第一章， 从Django1.6开始

指导你通过必要的基本配置以新建任意Django项目。本章覆盖内容有，虚拟环境，会话控制，以及项目设置。

第二章，数据库结构

教会你如何写可重复使用的代码片段并用在模型中。当你创建一个新的app时，要做的第一件就是定义模型。你也告知如何使用South迁移管理数据库表变更。

第三章，表单和视图

向你演示使用一些模式为数据创建视图和表单

第四章，模板和JavaScript

向你演示把模板和JavaScript放在一起使用的实际例子。我们把模板和JavaScript放在一起是因为，总是通过渲染模板将内容展现给用户，在现代的网站中，JavaScript对于更丰富的用户体验也是必要的。

第五章，自定义模板过滤器和标签

本章向你演示如如何创建并使用模板过滤器和标签，因为，Django的模板系统包含内容极广，因此可以有更多的东西对不同的应用场景来添加。

第六章，模型管理

本章，将指导你通过使用自定义的功能来扩展默认admin，

第七章，Django CMS

第八章，分层结构

第九章，数据的导入和导出

第十章，附加功能

扉页

章节预览 第二版

1. Getting Started with Django 1.8

- Introduction
- Working with a virtual environment
    - Getting ready
    - How to do it…
    - How it works…
    - See also
- Creating a project file structure
    - Getting ready
    - How to do it…
    - How it works…
    - See also
- Handling project dependencies with pip — Getting ready — How to do it… — How it works… — There's more… — See also — Making your code compatible with both Python 2.7 and Python 3 — Getting ready — How to do it… — How it works… — Including external dependencies in your project — Getting ready — How to do it… — How it works… — See also — Configuring settings for development, testing, staging, and production environments — Getting ready — How to do it… — How it works… — See also — Defining relative paths in the settings — Getting ready — How to do it… — How it works… — See also — Creating and including local settings — Getting ready — How to do it… —How it works… — See also — Setting up STATIC_URL dynamically for Subversion users —Getting ready — How to do it… — How it works… — See also — Setting up STATIC_URL dynamically for Git users —Getting ready — How to do it… — How it works… — See also — Setting UTF-8 as the default encoding for MySQL configuration — Getting ready —How to do it… —How it works… — Setting the Subversion ignore property — Getting ready — How to do it… — How it works… — See also — Creating the Git ignore file — Getting ready — How to do it… — How it works… — See also — Deleting Python-compiled files — Getting ready — How to do it… — How it works… — See also — Respecting the import order in Python files — Getting ready — How to do it… — How it works… — There's more… — See also — Creating app configuration — Getting ready — How to do it… — How it works… — There is

more… — See also
- Defining overwritable app settings — Getting ready — How to do it… — How it works…

2. Database Structure

Introduction Using model mixins Getting ready How to do it… How it works… There's more… See also Creating a model mixin with URL-related methods Getting ready How to do it… How it works… See also Creating a model mixin to handle creation and modification dates Getting ready How to do it… How it works… See also Creating a model mixin to take care of meta tags Getting ready How to do it… How it works… See also Creating a model mixin to handle generic relations Getting ready How to do it… How it works… See also Handling multilingual fields Getting ready How to do it… How it works… Using migrations Getting ready How to do it… How it works… See also Switching from South migrations to Django migrations Getting ready How to do it… How it works… See also Changing a foreign key to the many-to-many field Getting ready How to do it… How it works… See also

1. Forms and Views Introduction Passing HttpRequest to the form Getting ready How to do it… How it works… See also Utilizing the save method of the form Getting ready How to do it… How it works… See also Uploading images Getting ready How to do it… How it works… There's more See also Creating a form layout with django-crispy-forms Getting ready How to do it… How it works… There's more… See also Downloading authorized files Getting ready How to do it… How it works… See also Filtering object lists Getting ready How to do it… How it works… See also Managing paginated lists Getting ready How to do it… How it works… See also Composing class-based views Getting ready How to do it… How it works… There's more… See also Generating PDF documents Getting ready How to do it… How it works… See also Implementing a multilingual search with Haystack Getting ready How to do it… How it works… See also
2. Templates and JavaScript Introduction Arranging the base.html template Getting ready How to do it… How it works… See also Including JavaScript settings Getting ready How to do it… How it works… See also Using HTML5 data attributes Getting ready How to do it… How it works… See also Opening object details in a modal dialog Getting ready How to do it… How it works… See also Implementing a continuous scroll Getting ready How to do it… How it works… See also Implementing the Like widget Getting ready How to do it… How it works… See also Uploading images by Ajax Getting ready How to do it… How it works… See also
3. Custom Template Filters and Tags Introduction Following conventions for your own template filters and tags How to do it... Creating a template filter to show how many days have passed since a post was published Getting ready How to do it... How it works... There's more... See also Creating a template filter to extract the first media object Getting ready How to do it... How it works... There's more... See also Creating a

template filter to humanize URLs Getting ready How to do it... How it works... See also Creating a template tag to include a template if it exists Getting ready How to do it... How it works... There's more... See also Creating a template tag to load a QuerySet in a template Getting ready How to do it... How it works... See also Creating a template tag to parse content as a template Getting ready How to do it... How it works... See also Creating a template tag to modify request query parameters Getting ready How to do it... How it works... See also

4. Model Administration Introduction Customizing columns on the change list page Getting ready How to do it... How it works... There's more... See also Creating admin actions Getting ready How to do it... How it works... See also Developing change list filters Getting ready How to do it... How it works... See also Customizing default admin settings Getting ready How to do it... How it works... There's more... See also Inserting a map into a change form Getting ready How to do it... How it works... See also
5. Django CMS Introduction Creating templates for Django CMS Getting ready How to do it... How it works... See also Structuring the page menu Getting ready How to do it... How it works... See also Converting an app to a CMS app Getting ready How to do it... How it works... See also Attaching your own navigation Getting ready How to do it... How it works... See also Writing your own CMS plugin Getting ready How to do it... How it works... See also Adding new fields to the CMS page Getting ready How to do it... How it works... See also
6. Hierarchical Structures Introduction Creating hierarchical categories Getting ready How to do it... How it works... See also Creating a category administration interface with django-mptt-admin Getting ready How to do it... How it works... See also Creating a category administration interface with django-mptt-tree-editor Getting ready How to do it... How it works... See also Rendering categories in a template Getting ready How to do it... How it works... There's more... See also Using a single selection field to choose a category in forms Getting ready How to do it... How it works... See also Using a checkbox list to choose multiple categories in forms Getting ready How to do it... How it works... See also
7. Data Import and Export Introduction Importing data from a local CSV file Getting ready How to do it... How it works... There's more... See also Importing data from a local Excel file Getting ready How to do it... How it works... There's more... See also Importing data from an external JSON file Getting ready How to do it... How it works... See also Importing data from an external XML file Getting ready How to do it... How it works... There's more... See also Creating filterable RSS feeds Getting ready How to do it... How it works... See also Using Tastypie to create API Getting ready How to do it... How it works... See also Using Django REST framework to create API Getting ready How to do it... How it works... See also
8. Bells and Whistles Introduction Using the Django shell Getting ready How to do it... How it works... See also Using database query expressions Getting ready How to do it... How

it works... See also Monkey-patching the slugify() function for better internationalization support Getting ready How to do it... How it works... There's more... See also Toggling the Debug Toolbar Getting ready How to do it... How it works... See also Using ThreadLocalMiddleware Getting ready How to do it... How it works... See also Caching the method return value Getting ready How to do it... How it works... See also Using Memcached to cache Django views Getting ready How to do it... How it works... See also Using signals to notify administrators about new entries Getting ready How to do it... How it works... See also Checking for missing settings Getting ready How to do it... How it works... See also

9. Testing and Deployment Introduction Testing pages with Selenium Getting ready How to do it... How it works... See also Testing views with mock Getting ready How to do it... How it works... See also Testing API created using Django REST framework Getting ready How to do it... How it works... See also Releasing a reusable Django app Getting ready How to do it... How it works... See also Getting detailed error reporting via e-mail Getting ready How to do it... How it works... See also Deploying on Apache with mod_wsgi Getting ready How to do it... How it works... There's more... See also Setting up cron jobs for regular tasks Getting ready How to do it... How it works... See also Creating and using the Fabric deployment script Getting ready How to do it... How it works... There's more... See also

第一章， 从Django1.6开始

指导你通过必要的基本配置以新建任意Django项目。本章覆盖内容有，虚拟环境，会话控制，以及项目设置。

- 使用虚拟环境
- 创建一个项目文件结构
- 用pip处理项目依赖
- 在项目中包括外部的依赖
- 在settings中定义相对路径
- 为Subersion用户动态地配置STATIC_URL
- 为Git用户动态地配置STATIC_URL
- 创建并包括本地设置
- 把UTF-8设置为MySQL配置的默认编码格式
- 设置Subversion的忽略特性
- 创建Git的忽略文件
- 删除Python编译文件
- Python文件中的导入顺序
- 定义可重写的app设置

第二章，数据库结构

教会你如何写可重复使用的代码片段并用在模型中。当你创建一个新的app时，要做的第一件就是定义模型。你也告知如何使用Sou

- 使用模型mixin
- 使用相对URL方法创建一个模型mixin
- 创建一个模型mixin以处理日期的创建和修改
- 创建一个模型mixin以处理meta标签
- 创建一个模型mixin以处理通用关系
- 处理多语言字段
- 使用South迁移 （译者注：Django1.7中已经有了自己迁移模块，故内容将略去）
- 使用South将一个外键改变为多对多字段

第三章，表单和视图

向你演示使用一些模式为数据创建视图和表单

- 传递HttpRequest到表单
- 利用表单的save方法
- 上传图片
- 使用django-crispy-forms生成表单布局
- 过滤对象列表
- 管理分页列表
- 编写类视图
- 生成PDF文档

第四章，模板和JavaScript

向你演示把模板和JavaScript放在一起使用的实际例子。我们把模板和JavaScript放在一起是因为，总是通过渲染模板将内容展现给用户，在现代的网站中，JavaScript对于更丰富的用户体验也是必要的。

- 整理base.html模板
- 包含JavaScript设置
- 使用HTML5数据属性
- 在弹窗中显示对象细节
- 实现不间断滚动
- 实现Like部件
- 使用Ajax上传图片

第五章，自定义模板过滤器和标签

本章向你演示如如何创建并使用模板过滤器和标签，因为，Django的模板系统包含内容极广，因此可以有更多的东西对不同的应用场景来添加。

- 遵循模板过滤器和标签的约定
- 创建一个模板过滤器以显示经过的天数
- 创建一个模板过滤器提取第一个媒体对象
- 创建一个模板过滤器使URL可读
- 创建一个模板标签在模板中载入一个QuerySet
- 创建一个模板标签为模板解析内容
- 创建一个模板标签修改request查询参数

第六章，模型管理

本章，将指导你通过扩展默认管理带上自定义的功能，就和Django框架自带的预构建的模型管理一样好用。

- 定制换表页面中列
- 新建admin的行为
- 开发换表的过滤器
- 为外部的应用交换管理上的设置
- 将地图插入到交换表单

第七章，Django CMS

- 为Django CMS创建模板
- 组织页面按钮
- 将一个应用转换为CMS应用
- 添加自己的导航
- 编写自定义的CMS插件
- 对CMS页面添加新的字段

第八章，层级结构

- 生成层级目录
- 利用django-mptt-admin新建一个目录的管理接口
- 使用django-mptt-tree-editor创建一个目录的管理接口
- 在模板中渲染目录
- 在表单中利用一个单选字段来选择一个目录

- 于表单之中使用一个多选框列表来选择多个字段

第九章，数据的导入和导出

- 从本地的CSV文件中导入数据
- 由本地Excel文件导入数据
- 打外部JSON文件导入数据
- 自外部XML文件导入数据
- 创建可过滤的RSS订阅
- 使用Tastypie为第三方提供数据

第十章，附加功能

- 使用Django的命令行
- Using the Django shell
- The monkey patching slugification function
- The monkey patching model administration
- Toggling Debug Toolbar
- Using ThreadLocalMiddleware
- Caching the method value
- Getting detailed error reporting via e-mail
- Deploying on Apache with mod_wsgi
- Creating and using the Fabric deployment script

本章我们会学习到以下内容：

- 使用虚拟环境
- 创建一个项目文件结构
- 用pip处理项目依赖
- 在项目中包括外部的依赖
- 在settings中定义相对路径
- 为Subersion用户动态地配置STATIC_URL
- 为Git用户动态地配置STATIC_URL
- 创建并包括本地设置
- 把UTF-8设置为MySQL配置的默认编码格式
- 设置Subversion的忽略特性
- 创建Git的忽略文件
- 删除Python编译文件
- Python文件中的导入顺序
- 定义可重写的app设置

引言

为子版本用户动态地设置**STATIC_URL**

如果你对 `STATIC_URL` 设置一个静态值，那么每次你更新CSS文件，JavaScript文件，或者图片都需要清除浏览器的缓存以应用改变。有一个

预热

具体做法

实现原理

参见

为**Git**用户动态地设置**STATIC_URL**

预热

确保你的项目处理Git版本控制器之下。

具体做法

实现原理

参见

创建并包含本地设置

你不得不需要至少两个不同的项目实例：一个是创建新特性的开发环境，另一个是托管服务器中的公开网站环境。此外，可能会有针对其他开发者的不同的开发环境。你或许也需要有一个过渡性的环境以在一个类公开网站这样的情况下测试项目。

预热

不同环境的大多数设置都会共享并保存在版本控制中。不过，这里仍人会有一些针对某些项目环境的特别设置，例如，数据库或者电子邮件设置。我们把它们都放进 `local_settings.py` 文件中。

具体做法

执行以下步骤：

1. 在 `local_settings.py` 的尾部添加声明在相同目录下的 `local_settings.py` 的描述：

```
#settings.py
# ... 把这些代码放到文件的结尾 ...
try:
    execfile(os.path.join(os.path.dirname(__file__), ';local_settings'))
except IOError:
    pass
```

1. 创建 `local_settings.py` 然后把特定的环境设置加入到文件中：

```
#local_settings.py
DATABASES = {
    "default": {
        "ENGINE": "django.db.backends.mysql",
        "NAME": "myproject",
        "USER": "root",
        "PASSWORD": "root",
    }
    }
}
EMAIL_BACKEND = "django.core.mail.backends.console.EmailBackend"
INSTALLED_APPS += (
    "debug_toolbar",
)
```

工作原理

如你所见，本地设置并没有正常地导入，它们却包含在`settings.py`中并被执行。这样不仅允许你创建或者重写存在的设置，而且也可以调整`settings.py`文件中元组或者列表；例如，这里我们添加`debug_toolbar`到`INSTALLED_APPS`以启用对SQL查询，模板上下文变量，等等的调试。

参见

The Creating a project file structure recipe
The Toggling Debug Toolbar recipe in Chapter 10, Bells and Whistles

配置UTF_8作为MySQL配置的默认编码

预热

具体做法

工作原理

设置Subversion的忽略特性

预热

具体做法

打开命令行工具并社会默认编辑器为`nano，vi，vim`或者其他的任何你个人喜欢的编辑器：

```
$ export EDIOR=nano
```

> 提示
>
> 如果你还没有选择自己喜欢的编辑器，我推荐使用nano

工作原理

参见

创建Git的忽略文件

预热

具体做法

使用你最喜欢的文本编辑器，在Django项目的根目录下创建一个 `.gitinore` 文件，然后把这些文件和目录放进刚创建的文件中：

```
#.gitinore
*.pyc
/myproject/local_settings.py
/myproject/static/
/myproject/tmp/
/myproject/media/
```

工作原理

参见

The Setting the Subversion ignore property recipe

删除Python的编译文件

预热

具体做法

工作原理

参见

Python文件中的导入顺序

当你创建Python模块时，保持文件内的结构一致是个好的做法。这样做可以让其他的开发者和你自己在阅读代码时相对轻松一些。这个方法会向你演示如何组织导入。

预热

在Django项目中创建一个虚拟目录。

具体做法

在你创建的Python文件中应用下面的结构。然后要做的就是，在第一行定义UTF-8作为默认的Python文件编码，并把分类的导入放进文件区域：

```
# -*- coding: UTF-8 -*-
# System libraries
import os
import re
from datetime import datetime

# Third-party libraries
import boto
from PIL import Image

# Django modules
from django.db import models
from django.conf import settings

# Django apps
from cms.models import Page

# Current-app modules
import app_settings
```

工作原理

如下，我们有五个主要的目录被导入：

更多内容

参见

定义可重写的app设置

该做法会向你演示如何给应用定义设置，它可以在之后于项目的 `settings.py` 或者 `local_settings.py` 文件中被重写。对于可重复使用的应用该做法特别有效。

预热

手动地创建Django应用，或者利用下面的这个命令：

```
(myproject_env)$ django-admin.py startapp myapp1
```

具体做法

如果你刚好有一个到两个设置，你可以在 `models.py` 中使用下面的模式。如果设置也很多，你刚好也想要更好的组织它们，那么你可以在应用中创建一个文件 `app_settings.py` ，然后以下列方式写设置：

```
#models.py or app_settings.py
# -*- coding: UTF-8 -*-
from django.conf import settings
from django.utils.translation import ugettext_lazy as _

SETTING1 = getattr(settings, "MYAPP1_SETTING1", u"default value")
MEANING_OF_LIFE = getattr(settings, "MYAPP1_MEANING_OF_LIFE", 42)
STATUS_CHOICES = getattr(settings, "MYAPP1_STATUS_CHOICES", (
    ('draft', _("Draft")),
    ('published', _("Published")),
    ('not_listed', _("Not Listed")),
))
```

然后，你可以在 `models.py` 中以下面的方法使用应用的设置：

```
#models.py
# -*- coding: UTF-8 -*-
from django.db import models
from django.utils.translation import ugettext_lazy as _

from app_settings import STATUS_CHOICES

class NewsArticle(models.Model):
    # ...
    status = models.CharField(_("Status"),
        max_length=20, choices=STATUS_CHOICES
    )
```

如果你想要为一个项目重写 `STATUS_CHOICES` ，你只需简单地打开 `settings.py` 并加入下面代码：

```
#settings.py
# ...
from django.utils.translation import ugettext_lazy as _
MYAPP1_STATUS_CHOICES = (
    ("imported", _("Imported")),
    ("draft", _("Draft")),
    ("published", _("Published")),
    ("not_listed", _("Not Listed")),
    ("expired", _("Expired")),
)
```

工作原理

Python函数，`getattr(object, attribute_name[, default_value])`，视图从`object`获取属性`attribute_name`，如果未找到属性则返回`default_value`。这个例子中，不同的设置从Django项目设置模块中被视图取回，如果这些设置没找到，则默认值被使用。

第二章 数据库结构

本章节覆盖以下议题：

- 使用模型mixin
- 使用相对URL方法创建一个模型mixin
- 创建一个模型mixin以处理日期的创建和修改
- 创建一个模型mixin以处理meta标签
- 创建一个模型mixin以处理通用关系
- 处理多语言字段
- 使用South迁移 （译者注：Django1.7中已经有了自己迁移模块，故内容将略去）
- 使用South将一个外键改变为多对多字段

引言

当你新建新的app时，要做的第一件事就是创建表现数据库结构的模型。我们假设你之前已经创建了Django的app，要是没有话马上创建一个，而且你也阅读并了解Django的官方教程。本章，我会向你演示一些让数据库结构在项目的不同应用中保持一致的有趣的技术。然后我将向你演示创建模型字段以处理数据库中的数据的国际化。本章的最后，我会向你演示在开发的过程中如何使用迁移来改变数据库结构。

使用模型mixin

在Python这样的面向对象语言中，mixin类可以被视为一个实现功能的接口。当一个模型扩展了一个mixin，它就实现了接口，并包括了mixin的所有字段，特性，和方法。当你想要在不同的模型中重复地使用通用功能时，可以使用Django模型的mixin。

预热

要开始的话，你需要创建一些可重复使用的mixin。mixin的某些典型例子会在后面章节展示。一个保存模型mixin的好地方就是`utils`模块。

> 提示
>
> 如果你要创建一个与他人共享的重复使用app，那就要把模型mixin放在app里，比如放在应用的`base.py`文件中。

具体做法

在任何想要使用的mixin的Django应用中，创建 `models.py` 文件，并输入下面的代码：

```
#demo_app/models.py
# -*- coding: UTF-8 -*-
from django.db import models
from django.utils.translation import ugettext_lazy as _
from utils.models import UrlMixin
from utils.models import CreationModificationMixin
from utils.models import MetaTagsMixin

class Idea(UrlMixin, CreationModificationMixin, MetaTagsMixin):
    title = models.CharField(_("Title"), max_length=200)
    content = models.TextField(_("Content"))

    class Meta:
        verbose_name = _("Idea")
        verbose_name_plural = _("Ideas")

    def __unicode__(self):
        return self.title
```

工作原理

Django的模型继承支持三种类型的继承：抽象基类，多重继承，以及代理模型。模型mixin是拥有特定字段，属性，和方法的抽象模型类。当你创建前面的例子所示 `Idea` 这样的模型时，它从 `UrlMixin` ，`CreationModificationMixin` 和 `MetaTagsMixin` 继承了所有功能。所有的抽象类字段都作为所扩展模型的字段被保存在相同的数据库表中。

还有更多呢

为了学习更多不同类型的模型继承，参考Django官方文档 https://docs.djangoproject.com/en/dev/topics/db/models/#model-inheritance。

参见

- 使用相对URL方法创建一个模型mixin技巧
- 创建模型mixin以处理日期的创建和修改
- 创建模型mixin以处理meta标签

使用相对URL方法创建一个模型mixin

每个模型都有自己的页面，定义 `get_absolute_url()` 方法是很好的做法。这个方法可以用在模板中，它也可以用在Django admin站点中以预览所保存的项目。然而， `get_absolute_url` 很不明确，因为它实际上返回的是URL路径而不是完整的URL。在这个做法，我会向你演示如何创建一个模型mixin，默认它允许你定义URL路径或者完整的URL，以生成一个开箱即用的URL，并处理 `get_absolute_url` 方法的设置事宜。

预备！

如果你还没有完成创保存mixin的 `utils` 包。然后，在 `utils` 包内（可选的是，如果创建了一个重复使用的应用，那么你需要把 `base.py` 放在应用中）创建 `models.py` 文件。

具体做法

按步骤地执行以下命令：

1. 在 `utils` 包的 `models.py` 文件中添加以下内容：

```
#utils/models.py
# -*- coding: UTF-8 -*-
import urlparse
from django.db import models
from django.contrib.sites.models import Site
from django.conf import settings


class UrlMixin(models.Model):
   """
    替换get_absolute_url()。模型扩展该mixin便可以执行get_url或者get_url_path。
    """
    class Meta:
        abstract = True

    def get_url(self):
        if hasattr(self.get_url_path, "dont_recurse"):
            raise NotImplementedError
        try:
            path = self.get_url_path()
        except NotImplementedError:
            raise
        website_url = getattr(settings, "DEFAULT_WEBSITE_URL", "http://127.0.0.1:8000")
        return website_url + path
    get_url.dont_recurse = True

    def get_url_path(self):
        if hasattr(self.get_url, "dont_recurse"):
            raise NotImplementedError
        try:
            url = self.get_url()
        except NotImplementedError:
            raise
        bits = urlparse.urlparse(url)
        return urlparse.urlunparse(("", "") + bits[2:])
    get_url_path.dont_recurse = True

    def get_absolute_url(self):
        return self.get_url_path()
```

1. 为了在应用中使用mixin，需要把它从 `utils` 包导入，然后在模型类中继承mixin，并定义 `get_absolute_url()` 方法如下：

```
# demo_app/models.py
# -*- coding: UTF-8 -*-
from django.db import models
from django.utils.translation import ugettext_lazy as _
from django.core.urlresolvers import reverse
from utils.models import UrlMixin


class Idea(UrlMixin):
    title = models.CharField(_("Title"), max_length=200)
    #...

    def get_url_path(self):
        return reverse("idea_details", kwargs={
            "idea_id": str(self.pk)
            })
```

1. 如果你在临时或者生产环境中检查该代码，抑或你在本地运行不同的IP或者端口的服务器，那么你需要在本地设置的 `DEFAULT_WEBSITE_URL` 中设置如下内容：

```
#settings.py
# ...
DEFAULT_WEBSITE_URL = "http://www.example.com"
```

工作原理

`UrlMixin` 是一个拥有三种方法的抽象模型：`get_url()`，get_url_path()，get_absolute_url。get_url() 或者 get_url_path() 方法期待在所扩展的模型类中被重 Idea 类。你可定义 get_url，它是一个到对象的完整URL，get_url_path 会把它剥离到路径。你也可以定义 get_url_path，它是到对象的绝对路径，然后 get_url 会添加网站的URL到路径的开始。get_absolute_url 方法会 get_url_path`。

> ## 提示
>
> 通常的经验总是重写 `get_url_path()` 方法。
>
> 当你在同一个网站中需要到一个对象的链接，可以在模板中，使用 `<a href=""></a>`。对于电子邮件，RSS订阅或者API可以使用，`<a href=">"</a>`。

参阅

```
使用模型mixin

创建处理数据生成和修改的模型mixin

创建模型mixin以处理元标签

创建模型mixin处理通用关系
```

创建处理数据生成和修改的模型mixin

在模型中对于模型实例的创建和修改来说，一种常见的行为就是拥有时间戳。该方法中，我会向你演示如何给创建保存、修改模型的日期和时间。使用这样的mixin，可以保证所有的模型对于时间戳都使用相同的字段，以及拥有同样的行为。

准备开始

如果你还没有完成创保存mixin的 `utils` 包。然后，在 `utils` 包内（可选的是，如果创建了一个重复使用的应用，那么你需要把 `base.py` 放在应用中）创建 `models.py` 文件。

如何做

打开 `utils` 包中的 `models.py` 文件，并输入以下的代码：

```
#utils/models.py
# -*- coding: UTF-8 -*-
from django.db import models
from django.utils.translation import ugettext_lazy as _
from django.utils.timezone import now as timezone_now


class CreationModificationDateMixin(models.Model):
"""
可以创建和修改日期和时间的抽象基类。
"""
    created = models.DateTimeField(
        _("creation date and time"),
        editable=False,
    )

    modified = models.DateTimeField(
        _("modification date and time"),
        null=True,
        editable=False,
        )

    def save(self, *args, **kwargs):
        if not self.pk:
            self.created = timezone_now()
        else:
        #为了保证我们一直拥有创建数据，添加下面的条件语句
        if not self.created:
            self.created = timezone_now()
        self.modified = timezone_now()

        super(CreationModificationDateMixin, self).save(*args, **kwargs)
        save.alters_data = True

        class Meta:
            abstract = True
```

工作原理

`CreationModificationDateMixin` 类是一个抽象模型，这意味着它所扩展的模型类会在同一个数据表中创建所有字段，即，没有一对一关系让表变得难以处理。该mixin拥有两个日期-时间字段，以及一个在保存扩展模型会调用的 `save()` 方法。 `save()` 方法检查模型是否拥有主键，这是一种新建但还未保存的实例的情况。否则，如果主键存在，修改的日期就会被设置为当前的日期和时间。

作为选择，你可以不使用 `save()` 方法，而使用 `auto_now_add` 和 `auto_now` 属性来 `created` 和 `修改` 字段以自动地创建和修改时间戳。

参阅

使用模型mixin
创建模型mixin以处理meta标签
创建模型mixin以处理通用关系

创建模型mixin以处理meta标签

如果你想要为搜索引擎而优化网站，那么你不仅需要个每个页面都设置语义装饰，而且也需要合适的元标签。为了最大的灵活性，你需要有一种对在网站中有自己页面的所有对象都定义指定的元标签的方法。于此技法中，我们会向你演示如何对字段和方法创建一个mixin以关联到元标签。

准备开始喽！

和前面的做法一样，确保你为mixin备好了 `utils` 包。用你最喜欢的编辑器打开 `models.py` 文件。

具体做法

于 `models.py` 文件写入以下内容：

```
#utils/models.py
# -*- coding: UTF-8 -*-
from django.db import models
from django.utils.translation import ugettext_lazy as _
from django.template.defaultfilters import escape
from django.utils.safestring import mark_safe


class MetaTagsMixin(models.Model):
"""
用于<head>元素中由抽象基类所构成的元标签
"""
    meta_keywords = models.CharField(
        _("Keywords"),
        max_length=255,
        blank=True,
        help_text=_("Separate keywords by comma."),
    )
    meta_description = models.CharField(
        _("Description"),
        max_length=255,
        blank=True,
    )
    meta_author = models.CharField(
        _("Author"),
        max_length=255,
        blank=True,
    )
    meta_copyright = models.CharField(
        _("Copyright"),
        max_length=255,
        blank=True,
    )

    class Meta:
        abstract = True

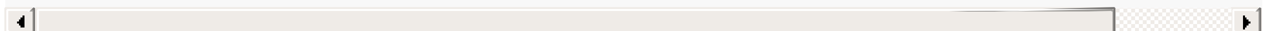
    def get_meta_keyword(self):
        tag = u" "
        if self.meta_keywords:
            tag = u'<meta name="keywords" content="{}".format(escape(self.meta_author)) /
            return mark_safe(tag)

    def get_meta_description(self):
        tag = u" "
        if self.meta_description:
            tag = u'<meta name="description" content="{1}".format(escape(self.meta_author
            return mark_safe(tag)

    def get_meta_author(self):
        tag = u" "
        if self.get_meta_author:
            tag = u'<meta name="author" content="{1}".format(escape(self.meta_author)) />

    def get_meta_copyright(self):
        tag = u" "
        if self.meta_copyright:
            tag = u'<meta name="copyright" content="{}".format(escape(self.meta_copyright
        return mark_safe(tag)

    def get_meta_tags(self):
        return mark_safe(u" ".join(
            self.get_meta_keyword(),
            self.get_meta_description(),
            self.get_meta_author(),
            self.get_meta_copyright(),
            ))
```



工作原理

该mixin添加了四个字段到模型扩展：`meta_keywords, meta_description, meta_author` 和 `meta_copyright`。在HTML中渲染元标签的方法也添加了。

如果你在 `Idea` 这样的模型中使用mixin，它出现在本章的第一个方法，那么你可以在目的是渲染所有元标签的详细页面模板的 `HEAD` 部分中写入以下内容：

```
{{ ieda.get_meta_tags }}
```

你也可以利用下面的行来渲染一个特定的元标签：

```
{{ idea.get_meta_description }}
```

或许你也注意到了代码片段，渲染的元标签被标记为安全，即，它们没有被转义而且我们也不要使用 `safe` 模板过滤器。只有来自数据库中的值才被转义，以保证最终的HTML成型良好。

参见

使用模型mixin
创建一个模型mixin以处理日期的创建和修改
创建一个模型mixin以处理通用关系

创建模型mixin以处理通用关系

除了外键关系或者对对关系这样的正常的数据库关系之外，Django还提供了一种关联一个模型到任意模型的实例。此概念称为通用关系。每个通用关系都有一个关联模型的内容类型，而且这个内容类型保存为该模型实例的ID。

该方法中，我们会向你演示如何将通用关系的创建归纳为模型mixin。

准备开始

要让该方法正常运行，你需要安装 `contenttypes` 应用。默认它应该位于 `INSTALLED_APPS` 目录中：

```
INSTALLED_APPS = (
    # ...
    "django.contrib.contenttypes",
)
```

再者，要确保你已经创建了放置模型mixin的`utils`包。

工作原理

在文本编辑器中打开`utils`包中的`models.py`文件，并输入以下的内容：

```
“
#utils/models.py
# -*- coding: UTF-8 -*-
from django.db import models
from django.utils.translation import ugettext_lazy as _
from django.contrib.contenttypes.models import ContentType
from django.contrib.contenttypes import generic
from django.core.exceptions import FieldError

def object_relation_mixin_factory(
    prefix=None,
    prefix_verbose=None,
    add_related_name=False,
    limit_content_type_choices_to={},
    limit_object_choices_to={},
    is_required=False,
)
""" 返回一个使用含有动态字段名称的“Content type - object Id”的mixin类的通用外键。
    该函数只是一个类生成器。
    参数：
        prefix：前缀用来添加到字段的前面
        prefix_verbose：前缀的冗余名称，用来生成Admin中内容对象的字段列的title
        add_related_name：表示一个布尔值，

"""
```

工作原理

参见

The Creating a model mixin with URL-related methods recipe
The Creating a model mixin to handle creation and modification dates recipe
The Creating a model mixin to take care of meta tags recipe

处理多语言字段

预热

具体做法

工作原理

使用South迁移

略。Django1.7已经自带迁移模块。

使用South将一个外键改变为多对多字段

略。

第三章-表单和视图

在本章，我们学习以下内容：

- 传递HttpRequest到表单
- 利用表单的save方法
- 上传图片
- 使用django-crispy-forms生成表单布局
- 过滤对象列表
- 管理分页列表
- 编写类视图
- 生成PDF文档

引言

当数据库结构定在模型中时，我们需要有一些视图供用户输入数据，或者对用户显示数据。本章，我们会关注管理表单的视图，列表视图，以及生成可替换的输出而不仅仅是HTML。举个最简单的例子来说，我们将把模板和URL规则的创建的决定权下放给你。

传递HttpRequest到表单

Django的每一个视图的第一个参数通常是命名为 `request` 的 `HttpRequest` 对象。它包含了请求的元数据，例如，当前语言编码，当前用户，当前cookie，或者是当前的session。默认，表单被用在视图中用来接受GET或者POST参数，文件，初始化数据，以及其他的参数，但却不是 `HttpRequest` 对象暗沟。某些情况下，特别是当你想要使用请求数据过滤出表单字段的选择，又或者是你想要处理像在表单中保存当前用户或者当前IP这样事情时，额外地传递地 `HttpRequest` 到表单会非常有用的。

在本方法中，我会向你展示一个在表单中某人可以选择一个用户并对此用户发送消息的例子。我们会传递 `HttpRequest` 对象到表单以暴露接受选项的当前用户：我们不想要任何人都可以给他们发送消息。

预热

让我们来创建一个叫做 `email_messages` 的应用，并把它放到设置中的 `INSTALLED_APPS` 去。该app仅含有表单和视图而没有模型。

具体做法

1. 添加一个新文件，`forms.py`，其中消息表单包含两个字段：接收人选项和消息文本。同时，该表单拥有一个初始化方法，它会接受 `request` 对象并对接收人的选项字段修改：

```
# -*- coding: UTF-8 -*-
from django import forms
from django.utils.translation import ugettext_lazy as _
from django.contrib.auth.models import User

class MessageForm(forms.Form):
    recipient = forms.ModelChoiceField(
        label=_("Recipient"),
        queryset=User.objects.all(),
        required=True,
    )
    message = forms.CharField(
        label=_("Message"),
        widget=forms.Textarea,
        required=True,
    )

    def __init__(self, request, *args, **kwargs):
        super(MessageForm, self).__init__(*args, **kwargs)
        self.request = request
        self.fields['recipient'].queryset = self.fields['recipient'].queryset.exclude(pk=
```

1. 然后，利用 `message_to_user` 视图创建 `views.py` 以处理表单。如你所见，`request` 对象作为第一个参数被传递到了表单：

```
#email_messages/views.py
# -*- coding: UTF-8 -*-
from django.contrib.auth.decorators import login_required
from django.shortcuts import render, redirect

from forms import MessageForm

@login_required
def message_to_user(request):
    if request.method == "POST":
        form = MessageForm(request, data=request.POST)
        if form.is_valid():
            # do something with the form
        return redirect("message_to_user_done")
    else:
        form = MessageForm(request)
    return render(request, "email_messages/message_to_user.html", {'form:form'})
```

工作原理

在初始化方法中，我们拥有表现表单自身实例的 `self` 变量，然后是新近添加的 `request` 变量，在然后是位置参数（`*args`）和命名参数（`**kwargs`）。我们调用 `super` 构造方法传递所有的位置参数以及命名参数，这样表单就可以正确地初始化。接着，我们把 `request` 变量赋值到一个新的表单的 `request` 变量，以便之后在表单的其他方法中可以访问。再然后，我们修改接收人选项字段的 `queryset` 属性以便从request暴露当前用户。

在视图中，我们将`HttpRequest`作为两种场合下的第一个参数来传递：当表单第一次载入时，以及表单发布之后。

参阅

表单的save方法的使用

表单的save方法使用

为了让视图变得简洁和简单，最好的做法是移动表单数据的处理到表单本身，不论是否可行。常见的做法是利用一个可以保存数据的save方法来执行搜索，或者来完成其他的复杂的行为。我们利用save方法扩展之前方法中所定义的表单，save方法会发送电子邮件到所选择的接收人。

预备开始

我们从定义在`传递HttpRequest到表单`这个例子开始。

具体做法

执行以下两步：

1. 在应用的表单中导入函数以便发送邮件。然后添加save方法并尝试发送邮件到所选择的接收人，发生错误时静默：

```
#email_messages/forms.py
# -*- coding: UTF-8 -*-
from django import forms
from django.utils.translation import ugettext, ugettext_lazy as _
from django.core.mail import send_mail
from django.contrib.auth.models import User

class MessageForm(forms.Form):
    recipient = forms.ModelChoiceField(
        label=_("Recipient"),
        queryset=User.objects.all(),
        required=True,
    )
    message = forms.CharField(
        label=_("Message"),
        widget=forms.Textarea,
        required=True,
    )

    def __init__(self, request, *args, **kwargs):
        super(MessageForm, self).__init__(*args, **kwargs)
        self.request = request
        self.fields['recipient'].queryset = \
            self.fields['recipient'].queryset.\
            exclude(pk=request.user.pk)

    def save(self):
        cleaned_data = self.cleaned_data
        send_mail(
            subject=ugettext("A message from %s") % \
                self.request.user,
            message=cleaned_data['message'],
            from_email=self.request.user.email,
            recipient_list=[cleaned_data['recipient'].email],
            fail_silently=True,
        )
```

1. 最后，假如发送的数据有效，则在视图中调用save方法：

```
#email_messages/views.py
# -*- coding: UTF-8 -*-
from django.contrib.auth.decorators import login_required
from django.shortcuts import render, redirect

from forms import MessageForm

@login_required
def message_to_user(request):
    if request.method == "POST":
        form = MessageForm(request, data=request.POST)
        if form.is_valid():
            form.save()
            return redirect("message_to_user_done")
    else:
        form = MessageForm(request)

    return render(request, "email_messages/message_to_user.html",{'form': form})
```

工作原理

首先让我们看下表单。save方法使用来自表单的清洁数据来读取接收人的电邮地址，以及电邮信息。电邮的发送人就是当前的request中的用户。假如电邮由于不正确的邮件服务器配置或者其他原因导致未能发送，那么错误也是静默的，即，不会在表单中抛出错误。

现在，我们来看下视图。当用户发送的表单有效时，表单的save方法会被调用，接着用户被重定向到成功页面。

参阅

传递HttpRequest到表单

上传图片

于此做法中，我们会看到处理图片上传的最简单的办法。你会见到一个应用中访客可以上传励志名言图片的例子。

预热

首先，让我们来创建一个应用 `quotes` ，并把它放到设置中的 `INSTALLED_APPS` 。然后，我们添加一个拥有三个字段的 `InspirationalQuote` 模型：作者，名言内容，以及图片，一如下面所示：

```
#quotes/models.py
#-*- coding:utf-8 -*-
import os
from django.db import models
from django.utils.timezone import now as timezone_now
from django.utils.translation impot ugetext_lazy as _


def upload_to(instance, filename):
    now = timezone_now()
    filename_base, filename_ext = os.path.splitext(filename)
    return 'quotes{}{}'.format(now.strftime("%Y/%m/%Y%m%d%H%M%S"), filename_ext.lower(),)

class InspirationalQuote(models.Model):
    author = models.CharField(_("Author"), max_length=200)
    quote = models.TextField(_("Quote"))
    picture = mdoels.ImageField(_("Picture"),
        upload_to=upload_to, blank=True, null=True,
        )

    class Meta:
        verbose_name = _("Inspirational Quote")
        verbose_name_plural = _("Inspiration Quotes")

    def __unicode__(self):
        return self.quote
```

此外，我们创建了一个函数 `upload_to` ,该函数设置类似 `quotes/2014/04/20140424140000` 这样的图片上传路径。你也看到了，我们使用日期时间戳作为文件名以确保文件的唯一性。我们传递该函数到 `picture` 图片字段。

具体做法

创建 `forms.py` 文件，并于其中编写一个简单的模型表单：

```
#quotes/forms.py
#-*- coding:utf-8 -*-
from django import forms
from models import InspirationQuote


class InspirationQuoteForm(forms.ModelForm):
    class Meta:
        model = InspirationQuote
```

在 `views.py` 文件中写入一个视图以处理表单。不要忘了传递类字典对象 `FILES` 到表单。如下，当表单有效时便会触发 `save` 方法：

```
#quotes/views.py
# -*- coding: UTF-8 -*-
from django.shortcuts import redirect
from django.shortcuts import render
from forms import InspirationQuoteForm


def add_quote(request):
    if request.method == 'POST':
        form = InspirationQuoteForm(
            data=request.POST,
            files=request.FIELS,
            )
        if form.is_valid():
            quote = form.save()
            return redirect("add_quote_done")
        else:
            form = InspirationQuoteForm()
        return render(request, "quotes/change_quote.html", {'form': form})
```

最后，在 `templates/quotes/change_quote.html` 中给视图创建一个模板。为HTML表单设置 `enctype` 属性为 `“multipart/form-data”` 十分的重要，否则文件上传不会起作用的：

```
{% extends "base.html" %}
{% load i18n %}

{% block content %}
    <form method="post" action="" enctype="multipart/form-data">
    {% csrf_token %}
        {{ form.as_p }}
        <button type="submit">{% trans "Save" %}
        </button>
    </form>
{% endblock %}
```

工作原理

Django的模型表单通过模型生成的表单。它们提供了所有的模型字段，因此你不要重复定义这些字段。前面的例子中，我们给`InspirationQuote`模型生成了一个模型表单。当我们保存表单时，表单知道如何包每个字段都保存到数据中去，也知道如何上传图片并在媒体目录中保存这些图片。

还有更多

作为奖励，我会想你演示一个如何从已经上传的图片中生成一个缩略图。利用这个技术，你也可以生成多个其他图片特定的版本，不如，列表版本，移动设备版本，桌面电脑版本。

我们添加三个方法到模型`InspirationQuote`（quotes/modles.py）。它们是save,create_thumbnail,和get_thumbnail_picture_url。当模型被保存时，我们就触发了缩略图的创建。如下，当我们需要在一个模板中显示缩略图时，我们可以通过使用`{{ quote.get_thumbnail_picture_url }}`来获取图片的URL：

```
#quotes/models.py
class InspirationQuote(models.Model):
    # ...
    def save(self, *args, **kwargs):
        super(InspirationQuote, self).save(*args, **kwargs)
        # 生成缩略图
        self.create_thumbnail()
    def create_thumbnail(self):
        from django.core.files.storage import default_storage as \
            storage
        if not self.picture:
            return ""
        file_path = self.picture.name
        filename_base, filename_ext = os.path.splitext(file_path)
        thumbnail_file_path = "%s_thumbnail.jpg" % filename_base
        if storage.exists(thumbnail_file_path):
            # if thumbnail version exists, return its url path
            # 如果缩略图存在，则返回缩略图的url路径
            return "exists"
        try:
            # resize the original image and
            # return URL path of the thumbnail version
            # 改变原始图片的大小并返回缩略图的URL路径
            f = storage.open(file_path, 'r')
            image = Image.open(f)
            width, height = image.size
            thumbnail_size = 50, 50

            if width > height:
                delta = width - height
                left = int(delta/2)
                upper = 0
                right = height + left
                lower = height
            else:
                delta = height - width
                left = 0
                upper = int(delta/2)
                right = width
                lower = width + upper

            image = image.crop((left, upper, right, lower))
            image = image.resize(thumbnail_size, Image.ANTIALIAS)

            f_mob = storage.open(thumbnail_file_path, "w")
            image.save(f_mob, "JPEG")
            f_mob.close()
            return "success"
        except:
            return "error"

    def get_thumbnail_picture_url(self):
        from PIL import Image
        from django.core.files.storage import default_storage as \
            storage
        if not self.picture:
            return ""
        file_path = self.picture.name
        filename_base, filename_ext = os.path.splitext(file_path)
        thumbnail_file_path = "%s_thumbnail.jpg" % filename_base
        if storage.exists(thumbnail_file_path):
            # if thumbnail version exists, return its URL path
            # 如果缩略图存在，则返回图片的URL路径
            return storage.url(thumbnail_file_path)
        # return original as a fallback
        # 返回一个图片的原始url路径
        return self.picture.url
```

之前的方法中，我们使用文件存储API而不是直接地与应对文件系统，因为我们之后可以使用亚马逊的云平台，或者其他的存储服务和方法来与默认的存储切换也可以正常工作。

缩略图的创建具体是如何工作的？如果原始图片保存为 `quotes/2014/04/20140424140000.png`，我们

参见

使用django-crispy-forms创建表单布局

使用django-crispy-forms创建表单布局

Django的应用，`django-crispy-forms` 允许你使用下面的CSS框架构建，定制，重复使用表单：`Uni-Form`，`Bootstrap`，或者 `Foundation`。django-crispy-form的用法类似于Django自带管理中的字段集合，而且它更高级，更富于定制化。按照Python代码定义表单布局，而且你不需要太过关心HTML中的每个字段。假如你需要添加指定的HTML属性或者指定的外观，你仍旧可以轻松地做到。

这个方法中，我们会向你演示一个如何使用拥有Bootstrap 3——它是开发响应式，移动设备优先的web项目的最流行的前端框架——的django-crispy-forms。

预热

我们一步接一步地执行这些步骤：

1.从http://getbootstrap.com/下载前端框架Bootstrap并将CSS和JavaScript集成到模板。更多内容详见第四章-模板和Javascript中的`管理base.html模板`。

2.使用下面的命令在虚拟环境中安装django-crispy-forms：

```
(myproject_env)$ pip install django-crispy-forms
```

3.确保csirpy-form添加到了 `INSTALLED_APPS`，然后在该项目中设置 `bootstrap3` 作为模板包来使用：

```
#settings.py
INSTALLED_APPS = (
    # ...
    "crispy_forms",
)
# ...
CRISPY_TEMPLATE_PACK = "bootstrap3"
```

4.让我们来创建一个应用 `bulletin_board` 来阐明django-crispy-forms的用法，并把应用添加到设置中的 `INSTALLED_APPS` 。我们会拥有一个含有这些字段的 `Bulletin` 模型：类型，名称，描述，联系人，电话，电邮，以及图片：

```
#bulletin_board/models.py
# -*- coding: UTF-8 -*-
from django.db import models
from django.utils.translation import ugettext_lazy as _

TYPE_CHOICES = (
    ('searching', _("Searching")),
    ('offering', _("Offering")),
)

class Bulletin(models.Model):
    bulletin_type = models.CharField(_("Type"), max_length=20, choices=TYPE_CHOICES)
    title = models.CharField(_("Title"), max_length=255)
    description = models.TextField(_("Description"),max_length=300)
    contact_person = models.CharField(_("Contact person"),
        max_length=255)
    phone = models.CharField(_("Phone"), max_length=200, blank=True)
    email = models.EmailField(_("Email"), blank=True)
    image = models.ImageField(_("Image"), max_length=255,
        upload_to="bulletin_board/", blank=True)

    class Meta:
        verbose_name = _("Bulletin")
        verbose_name_plural = _("Bulletins")
        ordering = ("title",)

    def __unicode__(self):
        return self.title
```

具体做法

Let's add a model form for the bulletin in the newly created app. We will attach a form helper to the form itself in the initialization method. The form helper will have the layout property, which will define the layout for the form, as follows:

```
#bulletin_board/forms.py
# -*- coding: UTF-8 -*-
from django import forms
from django.utils.translation import ugettext_lazy as _, ugettext
from crispy_forms.helper import FormHelper
from crispy_forms import layout, bootstrap
from models import Bulletin

class BulletinForm(forms.ModelForm):
    class Meta:
        model = Bulletin
        fields = ['bulletin_type', 'title', 'description',
            'contact_person', 'phone', 'email', 'image']

    def __init__(self, *args, **kwargs):
        super(BulletinForm, self).__init__(*args, **kwargs)

        self.helper = FormHelper()
        self.helper.form_action = ""
        self.helper.form_method = "POST"

        self.fields['bulletin_type'].widget = forms.RadioSelect()
        # delete empty choice for the type
        del self.fields['bulletin_type'].choices[0]

        self.helper.layout = layout.Layout(
            layout.Fieldset(
                _("Main data"),
                layout.Field("bulletin_type"),
                layout.Field("title", css_class="input-block-level"),
                layout.Field("description",
                    css_class="input-blocklevel", rows="3"),
            ),
            layout.Fieldset(
                _("Image"),
                layout.Field("image", css_class="input-block-level"),
                layout.HTML(u"""{% load i18n %}
                    <p class="help-block">{% trans "Available formats are JPG, GIF, and P
                """),
                title=_("Image upload"),
                css_id="image_fieldset",
            ),
            layout.Fieldset(
                _("Contact"),
                layout.Field("contact_person",
                     css_class="input-blocklevel"),
                layout.Div(
                    bootstrap.PrependedText("phone", """<span
class="glyphicon glyphicon-earphone"></span>""",
                        css_class="inputblock-level"),
                    bootstrap.PrependedText("email", "@",
                        css_class="input-block-level",
                        placeholder="contact@example.com"),
                    css_id="contact_info",
                ),
            ),
            bootstrap.FormActions(
                layout.Submit('submit', _('Save')),
            )
        )
```

要渲染模板中的表单，如下，我们只需载入标签冷酷 `crispy_forms_tags` ，然后使用模板标签 `{% crispy %}` ：

```
#templates/bulletin_board/change_form.html}
{% extends "base.html" %}
{% load crispy_forms_tags %}

{% block content %}
    {% crispy form %}
{% endblock %}
```

工作原理

拥有新闻简报表的页面的样子大概如此：

图片：略

As you see, the fields are grouped by fieldsets. The first argument of the Fieldset object defines the legend, the other positional arguments define fields. You can also pass named arguments to define HTML attributes for the fieldset; for example, for the second fieldset, we are passing title and css_id to set the HTML attributes title and id.

如你所见，字段是由字段集合组成的。

Fields can also have additional attributes passed by named arguments, for example, for the description field, we are passing css_class and rows to set the HTML attributes class and rows.

字段也可以通过传递命名参数拥有额外的属性，例如，

Besides the normal fields, you can pass HTML snippets as this is done with the help block for the image field. You can also have prepended-text fields in the layout, for example, we added a phone icon to the phone field, and an @ sign for the email field. As you see from the example with contact fields, we can easily wrap fields into HTML

elements using Div objects. This is useful when specific JavaScript needs to be applied to some form fields.

除了常规字段，你可以传递HTML片段

The action attribute for the HTML form is defined by the `form_action` property of the form helper. The method attribute of the HTML form is defined by the form_method property of the form helper. Finally, there is a Submit object to render the submit button, which takes the name of the button as the first positional argument, and the value of the button as the second argument.

还有更多

For the basic usage, the given example is more than necessary. However, if you need specific markup for forms in your project, you can still overwrite and modify templates of the `django-crispy-forms` app, as there is no markup hardcoded in Python files, but rather all the generated markup is rendered through the templates. Just copy the templates from the django-crispy-forms app to your project's template directory and change them as you need.

为了说明基本用法，给出例子是很有必要的一件事。不过，加入你需要在项目中为表单指定装饰，你仍然可以重写并修改`django-crispy-forms`这个应用的模板，在Python文件中不仅不存在由装饰的硬编码。

参阅

- 过滤对象列表
- 管理分页对象

过滤对象列表

在web开发中，除了视图和表单，拥有对象列表视图和详细视图是很典型的情况。列表视图可以简单的排列对象的顺序，例如，按找首字母排序或者创建日期来调用，不过对于非常庞大的数据来说就不是那么的友好了。

预备工作

For the filtering example, we will use the Movie model with relations to genres, directors, and actors to filter by. It will also be possible to filter by ratings, which is PositiveIntegerField with choices. Let's create the movies app, put it into INSTALLED_APPS in the settings (movies/models.py), and define the mentioned models in the new app:

为了说明过滤例子，我们会使用关联了种类、导演与演员的Moive模型进行过滤。而且通过评级过滤也是可以的，这是一个含有PositiveIntegerField的多选列表。我们来创建应用movies，并将它放到设置文件中的INSTALLED_APPS，然后在这个新应用中定义前面提及的模型：

```
#movies/models.py
# -*- coding: UTF-8 -*-
from django.db import models
from django.utils.translation import ugettext_lazy as _

RATING_CHOICES = (
    (1, u"☆"),
    (2, u"☆☆"),
    (3, u"☆☆☆"),
    (4, u"☆☆☆☆"),
    (5, u"☆☆☆☆☆"),
)

class Genre(models.Model):
    title = models.CharField(_("Title"), max_length=100)

    def __unicode__(self):
        return self.title

class Director(models.Model):
    first_name = models.CharField(_("First name"), max_length=40)
    last_name = models.CharField(_("Last name"), max_length=40)

    def __unicode__(self):
        return self.first_name + " " + self.last_name

class Actor(models.Model):
    first_name = models.CharField(_("First name"), max_length=40)
    last_name = models.CharField(_("Last name"), max_length=40)

    def __unicode__(self):
        return self.first_name + " " + self.last_name

class Movie(models.Model):
    title = models.CharField(_("Title"), max_length=255)
    genres = models.ManyToManyField(Genre, blank=True)
    directors = models.ManyToManyField(Director, blank=True)
    actors = models.ManyToManyField(Actor, blank=True)
    rating = models.PositiveIntegerField(choices=RATING_CHOICES)

    def __unicode__(self):
        return self.title
```

具体做法

首先，我们创建能够尽可能过滤所有目录的 `MovieFilterForm` ：

```
#movies/forms.py
# -*- coding: UTF-8 -*-
from django import forms
from django.utils.translation import ugettext_lazy as _

from models import Genre
from models import Director
from models import Actor
from models import RATING_CHOICES

class MovieFilterForm(forms.Form):
    genre = forms.ModelChoiceField(
        label=_("Genre"),
        required=False,
        queryset=Genre.objects.all(),
    )
    director = forms.ModelChoiceField(
        label=_("Director"),
        required=False,
        queryset=Director.objects.all(),
        )
    actor = forms.ModelChoiceField(
        label=_("Actor"),
        required=False,
        queryset=Actor.objects.all(),
    )
    rating = forms.ChoiceField(
        label=_("Rating"),
        required=False,
        choices=RATING_CHOICES,
    )
```

Then, we create a `movie_list` view that will use MovieFilterForm to validate the request query parameters and do the filtering by chosen categories. Note the facets dictionary, which is used here to list the categories and also the currently selected choices:

然后，我们创建一个使用MovieFilterForm验证请求查询参数的视图 `movie_list` ，再然后通过所选择的种类进行过滤。要注意字典这一方面，这里改字典用来列出目录以及当前选定的选项：

```
#movies/views.py
# -*- coding: UTF-8 -*-
from django.shortcuts import render
from models import Genre
from models import Director
from models import Actor
from models import Movie, RATING_CHOICES
from forms import MovieFilterForm

def movie_list(request):
    qs = Movie.objects.order_by('title')
    form = MovieFilterForm(data=request.REQUEST)

    facets = {
        'selected': {},
        'categories': {
            'genres': Genre.objects.all(),
            'directors': Director.objects.all(),
            'actors': Actor.objects.all(),
            'ratings': RATING_CHOICES,
        },
    }

    if form.is_valid():
        genre = form.cleaned_data['genre']
        if genre:
            facets['selected']['genre'] = genre
            qs = qs.filter(genres=genre).distinct()

        director = form.cleaned_data['director']
        if director:
            facets['selected']['director'] = director
            qs = qs.filter(directors=director).distinct()
            actor = form.cleaned_data['actor']
        if actor:
            facets['selected']['actor'] = actor
            qs = qs.filter(actors=actor).distinct()

        rating = form.cleaned_data['rating']
        if rating:
            facets['selected']['rating'] = (int(rating), dict(RATING_CHOICES)[int(rating)
            qs = qs.filter(rating=rating).distinct()

    context = {
        'form': form,
        'facets': facets,
        'object_list': qs,
    }
    return render(request, "movies/movie_list.html", context)
```

Lastly, we create the template for the list view. We will use the facets dictionary here to list the categories and to know which category is currently selected. To generate URLs for the filters, we will use the `{% append_to_query %}` template tag, which will be described later in the Creating a template tag to modify request query parameters recipe in Chapter 5, Custom Template Filters and Tags. Copy the following code in the templates/movies/movie_list.html directory:

最后，我们为列表视图创建模板。我们会使用

```
{#termplates/movies/movie_list.html}
{% extends "base_two_columns.html" %}
{% load i18n utility_tags %}

{% block sidebar %}
<div class="filters">
    <h6>{% trans "Filter by Genre" %}</h6>
    <div class="list-group">
        <a class="list-group-item{% if not facets.selected.genre %} active{% endif %}" hr
        {% for cat in facets.categories.genres %}
            <a class="list-group-item{% if facets.selected.genre == cat %} active{% endif
        {% endfor %}
    </div>

    <h6>{% trans "Filter by Director" %}</h6>
    <div class="list-group">
        <a class="list-group-item{% if not facets.selected.director %} active{% endif %}"
        href="{% append_to_query director="" page="" %}">{% trans "All" %}</a>
        {% for cat in facets.categories.directors %}
            <a class="list-group-item{% if facets.selected.director == cat %} active{% en
        {% endfor %}
    </div>

    <h6>{% trans "Filter by Actor" %}</h6>
    <div class="list-group">
        <a class="list-group-item{% if not facets.selected.actor %} active{% endif %}" hr
        {% for cat in facets.categories.actors %}
            <a class="list-group-item{% if facets.selected.actor == cat %} active{% endif
        {% endfor %}
    </div>

    <h6>{% trans "Filter by Rating" %}</h6>
    <div class="list-group">
        <a class="list-group-item{% if not facets.selected.rating %} active{% endif %}" h
        {% for r_val, r_display in facets.categories.ratings %}
            <a class="list-group-item{% if facets.selected.rating.0 == r_val %} active{%
        {% endfor %}
    </div>
</div>
{% endblock %}

{% block content %}
<div class="movie_list">
    {% for movie in object_list %}
        <div class="movie">
            <h3>{{ movie.title }}</h3>
        </div>
    {% endfor %}
</div>
{% endblock %}
```

工作原理

If we use the Bootstrap 3 frontend framework, the end result will look like this in the browser with some filters applied:

图片：略

So, we are using the facets dictionary that is passed to the template context, to know what filters we have and which filters are selected. To look deeper, the facets dictionary consists of two sections: the categories dictionary and the selected dictionary. The categories

dictionary contains the QuerySets or choices of all filterable categories. The selected dictionary contains the currently selected values for each category.

In the view, we check if the query parameters are valid in the form and then we drill down the QuerySet of objects by the selected categories. Additionally, we set the selected values to the facets dictionary, which will be passed to the template.

In the template, for each categorization from the facets dictionary, we list all categories and mark the currently selected category as active.

It is as simple as that.

参阅

The Managing paginated lists recipe

The Composing class-based views recipe

The Creating a template tag to modify request query parameters recipe in Chapter 5, Custom Template Filters and Tags

管理分页列表

If you have dynamically changing lists of objects or when the amount of them can be greater than 30-50, you surely need pagination for the list. With pagination instead of the full QuerySet, you provide a fraction of the dataset limited to a specific amount per page and you also show the links to get to the other pages of the list. Django has classes to manage paginated data, and in this recipe, I will show you how to do that for the example from the previous recipe.

预热

Let's start with the movies app and the forms as well as the views from the Filtering object lists recipe.

具体做法

At first, import the necessary pagination classes from Django. We will add pagination management to the `movie_list` view just after filtering. Also, we will slightly modify the context dictionary by passing a page instead of the movie QuerySet as `object_list` :

首先，从Django导入必需的分页类。我们会在过滤之后将分页管理添加到 `movie_list` 。而且，我们也要传递一个页面而不是movie的查询集合 `object_list` 来稍微修改上下文字典。

```
#movies/views.py
# -*- coding: UTF-8 -*-
from django.shortcuts import render
from django.core.paginator import Paginator, EmptyPage,\
    PageNotAnInteger

from models import Movie
from forms import MovieFilterForm

def movie_list(request):
    qs = Movie.objects.order_by('title')
    # ... filtering goes here...

    paginator = Paginator(qs, 15)

    page_number = request.GET.get('page')
    try:
        page = paginator.page(page_number)
    except PageNotAnInteger:
        # If page is not an integer, show first page.
        page = paginator.page(1)
    except EmptyPage:
        # If page is out of range, show last existing page.
        page = paginator.page(paginator.num_pages)

    context = {
        'form': form,
        'object_list': page,
    }
    return render(request, "movies/movie_list.html", context)
```

In the template, we will add pagination controls after the list of movies as follows:

```
{#templates/movies/movie_list.html#}
{% extends "base.html" %}
{% load i18n utility_tags %}

{% block sidebar %}
    {# ... filters go here... #}
{% endblock %}

{% block content %}
<div class="movie_list">
    {% for movie in object_list %}
        <div class="movie alert alert-info">
            <p>{{ movie.title }}</p>
        </div>
    {% endfor %}
</div>

{% if object_list.has_other_pages %}
    <ul class="pagination">
        {% if object_list.has_previous %}
            <li><a href="{% append_to_query page=object_list.previous_page_number %}">&la
        {% else %}
            <li class="disabled"><span>&laquo;</span></li>
        {% endif %}
        {% for page_number in object_list.paginator.page_range %}
            {% if page_number == object_list.number %}
                <li class="active">
                    <span>{{ page_number }} <span class="sr-only">(current)</span></span>
                </li>
            {% else %}
                <li>
                    <a href="{% append_to_query page=page_number %}">{{ page_number }}</a
                </li>
            {% endif %}
        {% endfor %}
        {% if object_list.has_next %}
            <li><a href="{% append_to_query page=object_list.next_page_number %}">&raquo;
        {% else %}
            <li class="disabled"><span>&raquo;</span></li>
        {% endif %}
    </ul>
{% endif %}

{% endblock %}
```

工作原理

When you look at the results in the browser, you will see pagination controls like these, added after the list of movies:

图片：略

How do we achieve that? When the QuerySet is filtered out, we create a paginator object passing the QuerySet and the maximal amount of items we want to show per page (which is 15 here). Then, we read the current page number from the query parameter, page. The next step is retrieving the current page object from the paginator. If the page number was not an integer, we get the first page. If the number exceeds the amount of possible pages, the last

page is retrieved. The page object has methods and attributes necessary for the pagination widget shown in the preceding screenshot. Also, the page object acts like a QuerySet, so that we can iterate through it and get the items from the fraction of the page.

The snippet marked in the template creates a pagination widget with the markup for the Bootstrap 3 frontend framework. We show the pagination controls only if there are more pages than the current one. We have the links to the previous and next pages, and the list of all page numbers in the widget. The current page number is marked as active. To generate URLs for the links, we are using the template tag {% append_to_query %}, which will be described later in the Creating a template tag to modify request query parameters recipe in Chapter 5, Custom Template Filters and Tags.

参阅

The Filtering object lists recipe
The Composing class-based views recipe
The Creating a template tag to modify request query parameters recipe in Chapter 5, Custom Template Filters and Tags

编写类视图

Django views are callables that take requests and return responses. In addition to function-based views, Django provides an alternative way to define views as classes. This approach is useful when you want to create reusable modular views or when you want to combine views out of generic mixins. In this recipe, we will convert the previously shown function-based view, movie_list, into a class-based view, MovieListView.

预热

Create the models, the form, and the template like in the previous recipes, Filtering object lists and Managing paginated lists.

具体做法

We will need to create a URL rule in the URL configuration and add a class-based view. To include a class-based view in the URL rules, the as_view()method is used like this:

```
#movies/urls.py
# -*- coding: UTF-8 -*-
from django.conf.urls import patterns, url
from views import MovieListView
urlpatterns = patterns('',
    url(r'^$', MovieListView.as_view(), name="movie_list"),
)
```

Our class-based view, MovieListView, will overwrite the get and post methods of the View class, which are used to distinguish between requests by GET and POST. We will also add the get_queryset_and_facets and get_page methods to make the class more modular:

```
#movies/views.py
# -*- coding: UTF-8 -*-
from django.shortcuts import render
from django.core.paginator import Paginator, EmptyPage,\
    PageNotAnInteger
from django.views.generic import View

from models import Genre
from models import Director
from models import Actor
from models import Movie, RATING_CHOICES
from forms import MovieFilterForm

class MovieListView(View):
    form_class = MovieFilterForm
    template_name = 'movies/movie_list.html'
    paginate_by = 15

    def get(self, request, *args, **kwargs):
        form = self.form_class(data=request.REQUEST)
        qs, facets = self.get_queryset_and_facets(form)
        page = self.get_page(request, qs)
        context = {
            'form': form,
            'facets': facets,
            'object_list': page,
        }
        return render(request, self.template_name, context)

    def post(self, request, *args, **kwargs):
        return self.get(request, *args, **kwargs)

    def get_queryset_and_facets(self, form):
        qs = Movie.objects.order_by('title')

        facets = {
            'selected': {},
            'categories': {
                'genres': Genre.objects.all(),
                'directors': Director.objects.all(),
                'actors': Actor.objects.all(),
                'ratings': RATING_CHOICES,
            },
        }
        if form.is_valid():
            genre = form.cleaned_data['genre']
            if genre:
                facets['selected']['genre'] = genre
                qs = qs.filter(genres=genre).distinct()

            director = form.cleaned_data['director']
            if director:
                facets['selected']['director'] = director
                qs = qs.filter(directors=director).distinct()
```

```
            actor = form.cleaned_data['actor']
            if actor:
                facets['selected']['actor'] = actor
                qs = qs.filter(actors=actor).distinct()

            rating = form.cleaned_data['rating']
            if rating:
                facets['selected']['rating'] = (int(rating),
                    dict(RATING_CHOICES)[int(rating)])
                qs = qs.filter(rating=rating).distinct()
        return qs, facets

    def get_page(self, request, qs):
        paginator = Paginator(qs, self.paginate_by)

        page_number = request.GET.get('page')
        try:
            page = paginator.page(page_number)
        except PageNotAnInteger:
            # If page is not an integer, show first page.
            page = paginator.page(1)
        except EmptyPage:
            # If page is out of range, show last existing page.
            page = paginator.page(paginator.num_pages)
        return page
```

工作原理

No matter whether the request was called by the GET or POST methods, we want the view to act the same; so, the post method is just calling the get method in this view, passing all positional and named arguments.

These are the things happening in the get method:

```
At first, we create the form object passing the REQUEST dictionary-like object to it. The

Then, the form is passed to the get_queryset_and_facets method, which respectively return

Then, the current request object and the QuerySet is passed to the get_page method, which

Lastly, we create a context dictionary and render the response.
```

还有更多

As you see, the get, post, and get_page methods are quite generic, so that we could create a generic class, FilterableListView, with those methods in the utils app. Then in any app, which needs a filterable list, we could create a view that extends FilterableListView and defines only the form_class and template_name attributes and the get_queryset_and_facets method. This is how class-based views work.

参阅

The Filtering object lists recipe
The Managing paginated lists recipe

生成PDF文档

Django views allow you to create much more than just HTML pages. You can generate files of any type. For example, you can create PDF documents for invoices, tickets, booking confirmations, or some other purposes. In this recipe, we will show you how to generate resumes (curriculum vitae) in PDF format out of the data from the database. We will be using the Pisa xhtml2pdf library, which is very practical as it allows you to use HTML templates to make PDF documents.

预热

First of all, we need to install the Python libraries reportlab and xhtml2pdf in your virtual environment:

```
(myproject_env)$ pip install reportlab==2.4
(myproject_env)$ pip install xhtml2pdf
```

Then, let us create a cv app with a simple CV model with the Experience model attached to it through a foreign key. The CV model will have these fields: first name, last name, and e-mail. The Experience model will have these fields: the start date at a job, the end date at a job, company, position at that company, and skills gained:

```
#cv/models.py
# -*- coding: UTF-8 -*-
from django.db import models
from django.utils.translation import ugettext_lazy as _

class CV(models.Model):
    first_name = models.CharField(_("First name"), max_length=40)
    last_name = models.CharField(_("Last name"), max_length=40)
    email = models.EmailField(_("Email"))

    def __unicode__(self):
        return self.first_name + " " + self.last_name

class Experience(models.Model):
    cv = models.ForeignKey(CV)
    from_date = models.DateField(_("From"))
    till_date = models.DateField(_("Till"), null=True, blank=True)
    company = models.CharField(_("Company"), max_length=100)
    position = models.CharField(_("Position"), max_length=100)
    skills = models.TextField(_("Skills gained"), blank=True)

    def __unicode__(self):
        till = _("present")
        if self.till_date:
            till = self.till_date.strftime('%m/%Y')
        return _("%(from)s-%(till)s %(position)s at %(company)s") % {
            'from': self.from_date.strftime('%m/%Y'),
            'till': till,'position': self.position,
            'company': self.company,
        }
    class Meta:
        ordering = ("-from_date",)
```

工作原理

In the URL rules, let us create a rule for the view to download a PDF document of a resume by the ID of the CV model, as follows:

```
#cv/urls.py
# -*- coding: UTF-8 -*-
from django.conf.urls import patterns, url

urlpatterns = patterns('cv.views',
    url(r'^(?P<cv_id>\d+)/pdf/$', 'download_cv_pdf',
    name='download_cv_pdf'),
)
```

Now let us create the download_cv_pdf view. This view renders an HTML template and then passes it to the PDF creator pisaDocument:

```
#cv/views.py
# -*- coding: UTF-8 -*-
try:
    from cStringIO import StringIO
except ImportError:
    from StringIO import StringIO
from xhtml2pdf import pisa

from django.conf import settings
from django.shortcuts import get_object_or_404
from django.template.loader import render_to_string
from django.http import HttpResponse

from cv.models import CV

def download_cv_pdf(request, cv_id):
    cv = get_object_or_404(CV, pk=cv_id)

    response = HttpResponse(mimetype='application/pdf')
    response['Content-Disposition'] = 'attachment; ' \
        'filename=%s_%s.pdf' % (cv.first_name, cv.last_name)

    html = render_to_string("cv/cv_pdf.html", {
        'cv': cv,
        'MEDIA_ROOT': settings.MEDIA_ROOT,
        'STATIC_ROOT': settings.STATIC_ROOT,
    })

    pdf = pisa.pisaDocument(
        StringIO(html.encode("UTF-8")),
        response,
        encoding='UTF-8',
    )
    return response
```

At last, we create the template by which the document will be rendered, as follows:

```
{#templates/cv/cv_pdf.html#}
<!DOCTYPE HTML>
<html>
    <head>
        <meta charset="utf-8" />
        <title>My Title</title>
        <style type="text/css">
            @page {
                size: "A4";
                margin: 2.5cm 1.5cm 2.5cm 1.5cm;
                @frame footer {
                    -pdf-frame-content: footerContent;
                    bottom: 0cm;
                    margin-left: 0cm;
                    margin-right: 0cm;
                    height: 1cm;
                }
            }
            #footerContent {
                color: #666;
                font-size: 10pt;
                text-align: center;
            }
            /* ... Other CSS Rules go here ... */

        </style>
    </head>
    <body>
        <div>
            <p class="h1">Curriculum Vitae</p>
            <table>
```

```
                <tr>
                    <td><p><b>{{ cv.first_name }} {{ cv.last_name }}</b><br />
                        Contact: {{ cv.email }}</p>
                    </td>
                    <td align="right">
                        <img src="{{ STATIC_ROOT }}/site/img/smiley.jpg" width="100" heig
                    </td>
                </tr>
            </table>

            <p class="h2">Experience</p>
            <table>
                {% for experience in cv.experience_set.all %}
                <tr>
                        <td><p>{{ experience.from_date|date:"F Y" }} -
                            {% if experience.till_date %}
                                {{ experience.till_date|date:"F Y" }}
                            {% else %}
                                present
                            {% endif %}<br />
                            {{ experience.position }} at {{ experience.company }}</p>
                        </td>
                        <td><p><b>Skills gained</b><br>
                            {{ experience.skills|linebreaksbr }}
                            <br>
                            <br>
                            </p>
                        </td>
                    </tr>
                {% endfor %}
            </table>
        </div>
        <pdf:nextpage>
        <div>
            This is an empty page to make a paper plane.
        </div>
        <div id="footerContent">
            Document generated at {% now "Y-m-d" %} |
            Page <pdf:pagenumber> of <pdf:pagecount>
        </div>
    </body>
</html>
```

工作原理

Depending on the data entered into the database, the rendered PDF document might look like this:

图片：略

How does the view work? At first, we load a curriculum vitae by its ID if it exists, or raise the Page-not-found error if it does not. Then, we create the response object with mimetype of the PDF document. We set the Content-Disposition header to attachment with the specified filename. This will force browsers to open a dialog box prompting to save the PDF document and will suggest the specified name for the file. Then, we render the HTML template as a string passing curriculum vitae object, and the MEDIA_ROOT and STATIC_ROOT paths.

> 提示
>
> Note that the src attribute of the tag used for PDF creation needs to point to the file in the filesystem or the full URL of the image online.

Then, we create a pisaDocument file with the UTF-8 encoded HTML as source, and response object as the destination. The response object is a file-like object, and pisaDocument writes the content of the document to it. The response object is returned by the view as expected.

Let us have a look at the HTML template used to create this document. The template has some unusual HTML tags and CSS rules. If we want to have some elements on each page of the document, we can create CSS frames for that. In the preceding example, the

tag with the footerContent ID is marked as a frame, which will be repeated at the bottom of each page. In a similar way, we can have a header or a background image for each page.

Here are the specific HTML tags used in this document:

```
The <pdf:nextpage> tag sets a manual page break
The <pdf:pagenumber> tag returns the number of the current page
The <pdf:pagecount> tag returns the total number of pages
```

The current version 3.0.33 of the Pisa xhtml2pdf library does not fully support all HTML tags and CSS rules; for example,

,

, and other headings are broken and there is no concept of floating, so instead you have to use paragraphs with CSS classes for different font styles and tables for multi-column layouts. However, this library is still mighty enough for customized layouts, which basically can be created just with the knowledge of HTML and CSS.

参阅

The Managing paginated lists recipe

本章完

第四章-模板和JavaScript

本章，我们讨论到以下话题：

```
整理base.html模板
包含JavaScript设置
使用HTML5数据属性
在弹窗中显示对象细节
实现不间断滚动
实现Like部件
使用Ajax上传图片
```

引言

我们生活在

整理base.html模板

> 提示

预热

具体做法

执行以下步骤：

1.在模板的根目录中使用下列内容创建一个`base.html`文件：

```
{#templates/base.html#}
{% block doctype %}<!DOCTYPE html>{% endblock %}
{% load i18n %}
<html lang="{{ LANGUAGE_CODE }}">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <title>{% block title %}{% endblock %}{% trans "My Website" %}</title>
    <link rel="icon" href="{{ STATIC_URL }}site/img/favicon.ico" type="image/png" />

    {% block meta_tags %}{% endblock %}

    <link rel="stylesheet" href="http://netdna.bootstrapcdn.com/bootstrap/3.1.1/css/boots
    “ <link href="{{ STATIC_URL }}site/css/style.css" rel="stylesheet" media="screen" typ

    {% block stylesheet %}{% endblock %}
    <script src="http://code.jquery.com/jquery-1.11.0.min.js"></script>
    <script src="http://code.jquery.com/jquery-migrate-1.2.1.min.js"></script>
    <script src="http://netdna.bootstrapcdn.com/bootstrap/3.1.1/js/bootstrap.min.js"></sc
    <script src="{% url "js_settings" %}"></script>
    {% block js %}{% endblock %}
    {% block extrahead %}{% endblock %}
</head>
<body class="{% block bodyclass %}{% endblock %}">
    {% block page %}
        <div class="wrapper">
            <div id="header" class="clearfix">
                <h1>{% trans "My Website" %}</h1>
                {% block header_navigation %}
                    {% include "utils/header_navigation.html" %}
                {% endblock %}
                {% block language_chooser %}
                    {% include "utils/language_chooser.html" %}
                {% endblock %}
            </div>
            <div id="content" class="clearfix">
                {% block content %}
                {% endblock %}
            </div>
            <div id="footer" class="clearfix">
                {% block footer_navigation %}
                    {% include "utils/footer_navigation.html" %}
                {% endblock %}
            </div>
        </div>
    {% endblock %}
    {% block extrabody %}{% endblock %}
</body>
</html>
```

1. 在相同的目录中，针对特定的情况创建另外一个命名为 `base_simple.html` ：

```
{# templates/base_simple.html #}
{% extends "base.html" %}

{% block page %}
    <div class="wrapper">
        <div id="content" class="clearfix">
            {% block content %}
            {% endblock %}
        </div>
    </div>
{% endblock %}
```

具体做法

参阅

加入JavaScript的设置

第五章 定制模板过滤器和标签

本章，我们会学习以下内容：

- 遵循模板过滤器和标签的约定
- 创建一个模板过滤器显示已经过去的天数
- 创建一个模板过滤器提取第一个媒体对象
- 创建一个模板过滤器使URL可读
- 创建一个模板标签在模板中载入一个QuerySet
- 创建一个模板标签为模板解析内容
- 创建一个模板标签修改request查询参数

简介

如你所知，Django有一个非常庞大的模板系统，特别是模板继承，

遵循模板过滤器和标签的约定

Custom template filters and tags can become a total mess if you don't have persistent guidelines to follow. Template filters and tags should serve template editors as much as possible. They should be both handy and flexible. In this recipe, we will look at some conventions that should be used when enhancing the functionality of the Django template system.

滤器和标签时会让你完全不知所措。模板过滤器和标签应该尽可能的服务于模板编辑器，而且它们都应该同时具有操作方便和灵活扩展的特性。在这个做法中，我们会见到一些在增强Django模板系统功能时所用到的几点约定。

如何做

扩张Django模板系统时遵循以下约定：

```
1. 当在视图，上下文处理器，或者模型方法中，页面更适合逻辑时，不要创建或者使用定制模板过滤器、标签。你的页面

2. 使用 _tags后缀命名模板标签库。当你的app命名不同于模板标签库时，你可以避免模糊的包导入问题。

3. 例如，通过使用如下注释，在最后创建的库中，从标签中分离过滤器：

# -*- coding: UTF-8 -*-
from django import template
register = template.Library()
### FILTERS ###
# .. your filters go here ..

### TAGS ###
# .. your tags go here..

4. 通过下面的格式，创建的模板标签就可以被轻松记住：

    for [app_name.model_name]：使用该格式以使用指定的模型

    using [template_name]：使用该格式将一个模板的模板标签输出

    limit [count]：使用该格式将结果限制为一个指定的数量

    as [context_variable]：使用该结构可以将结构保存到一个在之后多次使用的上下文变量

5. 要尽量避免在模板标签中按位置地定义多个值除非它们都是不解自明的。否则，这会有可能使开发者迷惑。

6. 尽可能的使用更多的可理解的参数。没有引号的字符串应该当作需要解析的上下文变量或者可以提醒你关于模板标签组
```

参见

The Creating a template filter to show how many days have passed recipe The Creating a template filter to extract the first media object recipe The Creating a template filter to humanize URLs recipe The Creating a template tag to include a template if it exists recipe The Creating a template tag to load a QuerySet in a template recipe The Creating a template tag to parse content as a template recipe The Creating a template tag to modify request query parameters recipe”

创建一个模板过滤器显示已经过去的天数

Not all people keep track of the date, and when talking about creation or modification dates of cutting-edge information, for many of us, it is more convenient to read the time difference, for example, the blog entry was posted three days ago, the news article was published today, and the user last logged in yesterday. In this recipe, we will create a template filter named days_since that converts dates to humanized time differences.

不是所有的人都持续关注时间，而且在谈论最新日期信息的新建和修改时，对于我们大多数人来说读取时间差是更为合适的一种选择，例如，博客内容在发布于三天之前，新的文章在今天发布了，而用户则是在昨天进行了最后的登陆行为。在这个做法中，我们将新一个称作days_since的可以转换日期到人类可读时间差的模板过滤器。

准备开始

Create the utils app and put it under `INSTALLED_APPS` in the settings, if you haven't done that yet. Then, create a Python package named templatetags inside this app (Python packages are directories with an empty `__init__.py` file).
创建utils应用并将它放到settings文件中的 `INSTALLED_APPS` 下，要是你还没有按照我说的去做的话。然后，在这个应用里边创建一个名叫templatetags的Python文件夹（Python包是一个拥有内容为空的 `__init__.py` 文件）。

我该怎么做

使用下面的内容创建一个utility_tags.py文件：

```
#utils/templatetags/utility_tags.py
# -*- coding: UTF-8 -*-
from datetime import datetime

from django import template
from django.utils.translation import ugettext_lazy as _
from django.utils.timezone import now as tz_now
register = template.Library()

### FILTERS ###

@register.filter
def days_since(value):
    """ Returns number of days between today and value."""

    today = tz_now().date()
    if isinstance(value, datetime.datetime):
        value = value.date()
    diff = today - value
    if diff.days > 1:
        return _("%s days ago") % diff.days
    elif diff.days == 1:
        return _("yesterday")
    elif diff.days == 0:
        return _("today")
    else:
        # Date is in the future; return formatted date.
        return value.strftime("%B %d, %Y")
```

工作原理

If you use this filter in a template like the following, it will render something like yesterday or 5 days ago:

假如你在模板中使用了如下所示的过滤器，那么该过滤器会将日期渲染为比如，今天，或者五天前：

```
"{% load utility_tags %}
{{ object.created|days_since }}
You can apply this filter to the values of the date and datetime types.
```

Each template-tag library has a register where filters and tags are collected. Django filters are functions registered by the `register.filter` decorator. By default, the filter in the template system will be named the same as the function or the other callable object. If you want, you can set a different name for the filter by passing name to the decorator, as follows:

每个模板标签库都有一个收集过滤器和标签的注册器。Django过滤器是通过装饰器 `register.filter` 注册过的函数。默认，模板系统中的过滤器的名称和函数或者其他可调用对象的名称相同。如下，如果你有需要的话，你可以通过传递名称到装饰器来为过滤器设置一个不同的名称：

```
@register.filter(name="humanized_days_since")
def days_since(value):
    ...
```

The filter itself is quite self-explanatory. At first, the current date is read. If the given value of the filter is of the datetime type, the date is extracted. Then, the difference between today and the extracted value is calculated. Depending on the number of days, different string results are returned.

还有更多

This filter is easy to extend to also show the difference in time, such as just now, 7 minutes ago, or 3 hours ago. Just operate the datetime values instead of the date values.

参阅

The Creating a template filter to extract the first media object recipe
The Creating a template filter to humanize URLs recipe

创建一个模板过滤器以提取第一个媒体对象

Imagine that you are developing a blog overview page, and for each post, you want to show images, music, or videos in that page taken from the content. In such a case, you need to extract the `<img>`, `<object>`, and `<embed>` tags out of the HTML content of the post. In this recipe, we will see how to do this using regular expressions in the `get_first_media` filter.

准备开始吧

We will start with the `utils` app that should be set in `INSTALLED_APPS` in the settings and the `templatetags` package inside this app.

如何做

In the `utility_tags.py` file, add the following content:
在 `utility_tags.py` 文件中，添加以下内容：

```
#utils/templatetags/utility_tags.py
# -*- coding: UTF-8 -*-
import re
from django import template
from django.utils.safestring import mark_safe
register = template.Library()

### FILTERS ###

media_file_regex = re.compile(r"<object .+?</object>|"
    r"<(img|embed) [^>]+>") )

@register.filter
def get_first_media(content):
    """ Returns the first image or flash file from the html content """
    m = media_file_regex.search(content)
    media_tag = ""
    if m:
        media_tag = m.group()
    return mark_safe(media_tag)
```

工作原理

While the HTML content in the database is valid, when you put the following code in the template, it will retrieve the `<object>`, `<img>`, or `<embed>` tags from the content field of the object, or an empty string if no media is found there:

```
{% load utility_tags %}
{{ object.content|get_first_media }}
```

At first, we define the compiled regular expression as `media_file_regex`, then in the filter, we perform a search for that regular expression pattern. By default, the result will show the <, >, and & symbols escaped as `<`, `>`, and `&amp` ; entities. But we use the `mark_safe` function that marks the result as safe HTML ready to be shown in the template without escaping.

还有更多

It is very easy to extend this filter to also extract the `<iframe>` tags (which are more recently being used by Vimeo and YouTube for embedded videos) or the HTML5 `<audio>` and `<video>` tags. Just modify the regular expression like this:

```
media_file_regex = re.compile(r"<iframe .+?</iframe>|"
    r"<audio .+?</ audio>|<video .+?</video>|"
    r"<object .+?</object>|<(img|embed) [^>]+>") ″
```

参阅

The Creating a template filter to show how many days have passed recipe

The Creating a template filter to humanize URLs recipe

创建模板过滤器以人性化URL

Usually, common web users enter URLs into address fields without protocol and trailing slashes. In this recipe, we will create a humanize_url filter used to present URLs to the user in a shorter format, truncating very long addresses, just like what Twitter does with the links in tweets.

准备开始

As in the previous recipes, we will start with the `utils` app that should be set in `INSTALLED_APPS` in the settings, and should contain the `templatetags` package.

就像之前技巧中提到的，我们会从设置于 `INSTALLED_APPS` 中的 `utils` 应用开始，它应该包含一个 `templatetags` 包。

如何做

In the FILTERS section of the utility_tags.py template library in the utils app, let's add a filter named humanize_url and register it:

```
#utils/templatetags/utility_tags.py
# -*- coding: UTF-8 -*-
import re
from django import template
register = template.Library()

### FILTERS ###

@register.filter
def humanize_url(url, letter_count):
    """ Returns a shortened human-readable URL """
    letter_count = int(letter_count)
    re_start = re.compile(r"^https?://")
    re_end = re.compile(r"/$")
    url = re_end.sub("", re_start.sub("", url))
    if len(url) > letter_count:
        url = u"%s…" % url[:letter_count - 1]
    return url
```

工作原理

We can use the humanize_url filter in any template like this:

```
{% load utility_tags %}
<a href="{{ object.website }}" target="_blank">
    {{ object.website|humanize_url:30 }}
</a>
```

The filter uses regular expressions to remove the leading protocol and the trailing slash, and then shortens the URL to the given amount of letters, adding an ellipsis to the end if the URL doesn't fit into the specified letter count.

参见

The Creating a template filter to show how many days have passed recipe
The Creating a template filter to extract the first media object recipe
The Creating a template tag to include a template if it exists recipe

如果存在一个模板，创建一个模板标签以包含它

Django has the `{% include %}` template tag that renders and includes another template. However, in some particular situations, there is a problem that an error is raised if the template does not exist. In this recipe, we will show you how to create a `{% try_to_include %}` template tag that includes another template, but fails silently if there is no such template.

准备开始

We will start again with the utils app that should be installed and is ready for custom template tags.

如何做

Template tags consist of two things: the function parsing the arguments of the template tag and the node class that is responsible for the logic of the template tag as well as for the output. Perform the following steps:

模板标签由两个东西组成：解析模板标签参数对函数，以及能够良好输出的响应模板标签逻辑的节点类。记下来执行以下步骤：

1. First, let's create the function parsing the template-tag arguments:
   首先，让我们来创建解析模板标签参数的函数：

```
#utils/templatetags/utility_tags.py
# -*- coding: UTF-8 -*-
from django import template
from django.template.loader import get_template
register = template.Library()

### TAGS ###

@register.tag
def try_to_include(parser, token):
    """Usage: {% try_to_include "sometemplate.html" %}
    This will fail silently if the template doesn't exist.
    If it does, it will be rendered with the current context."""
    try:
        tag_name, template_name = token.split_contents()
    except ValueError:
        raise template.TemplateSyntaxError, \
            "%r tag requires a single argument" % \token.contents.split()[0]
    return IncludeNode(template_name)
```

1. Then, we need the node class in the same file, as follows:

```
class IncludeNode(template.Node):
    def __init__(self, template_name):
        self.template_name = template_name

    def render(self, context):
        try:
            # Loading the template and rendering it
            template_name = template.resolve_variable(
                self. template_name, context)
            included_template = get_template(template_name).\
                render(context)
        except template.TemplateDoesNotExist:
            included_template = ""
        return included_template
```

The `{% try_to_include %}` template tag expects one argument, that is, `template_name`. So, in the `try_to_include` function, we are trying to assign the split contents of the token only to the `tag_name` variable (which is `"try_to_include"`) and the `template_name` variable. If this doesn't work, the template syntax error is raised. The function returns the `IncludeNode` object, which gets the `template_name` field for later usage.

`{% try_to_include %}`模板标签需要一个参数，即，`template_name`。因此，在函数`try_to_include`中，

In the render method of `IncludeNode`, we resolve the `template_name` variable. If a context variable was passed to the template tag, then its value will be used here for `template_name`. If a quoted string was passed to the template tag, then the content within quotes will be used for `template_name`.

Lastly, we try to load the template and render it with the current template context. If that doesn't work, an empty string is returned.

There are at least two situations where we could use this template tag:

```
When including a template whose path is defined in a model, as follows:
```

```
{% load utility_tags %}
{% try_to_include object.template_path %}
```

```
When including a template whose path is defined with the {% with %} template     tag som
```

```
#templates/cms/start_page.html
{% with editorial_content_template_path="cms/plugins/editorial_content/start_page.html" %
    {% placeholder "main_content" %}
{% endwith %}

#templates/cms/plugins/editorial_content.html
{% load utility_tags %}

{% if editorial_content_template_path %}
    {% try_to_include editorial_content_template_path %}
{% else %}
    <div>
        <!-- Some default presentation of
             editorial content plugin -->
   </div>
{% endif %}
```

还有更多

You can use the `{% try_to_include %}` tag as well as the default `{% include %}` tag to include templates that extend other templates. This has a beneficial use for large-scale portals where you have different kinds of lists in which complex items share the same structure as widgets but have a different source of data.

For example, in the artist list template, you can include the artist item template as follows:

```
{% load utility_tags %}
{% for object in object_list %}
    {% try_to_include "artists/includes/artist_item.html" %}
{% endfor %}
```

This template will extend from the item base as follows:

```
{# templates/artists/includes/artist_item.html #}
{% extends "utils/includes/item_base.html" %}

{% block item_title %}
    {{ object.first_name }} {{ object.last_name }}
{% endblock %}
```

The item base defines the markup for any item and also includes a Like widget, as follows:

```
{# templates/utils/includes/item_base.html #}
{% load likes_tags %}

<h3>{% block item_title %}{% endblock %}</h3>
{% if request.user.is_authenticated %}
    {% like_widget for object %}
{% endif %}
```

参阅

The Creating templates for Django CMS recipe in Chapter 7, Django CMS
The Writing your own CMS plugin recipe in Chapter 7, Django CMS
The Implementing a Like recipe in Chapter 4, Templates and JavaScript
The Creating a template tag to load a QuerySet in a template recipe
The Creating a template tag to parse content as a template recipe
The Creating a template tag to modify request query parameters recipe

创建一个模板标签以载入模板中的Queryset

Most often, the content that should be shown in a web page will have to be defined in the view. If this is the content to show on every page, it is logical to create a context processor. Another situation is when you need to show additional content such as the latest news or a

random quote on some specific pages, for example, the start page or the details page of an object. In this case, you can load the necessary content with the `{% get_objects %}` template tag, which we will implement in this recipe.

准备开始

Once again, we will start with the `utils` app that should be installed and ready for custom template tags.

我们会再一次从安装好的`utils`应用开始，为定制模板标签做好准备。

如何做

Template tags consist of function parsing arguments passed to the tag and a node class that renders the output of the tag or modifies the template context. Perform the following steps:

模板标签由解析传递到标签的参数的函数组成，而节点类会传递外部的标签或者修改模板上下文。要达成模板，需执行以下步骤：

1. First, let's create the function parsing the template-tag arguments, as follows:
   首选，让我们来创建函数去解析模板标签参数，一如下面内容所示：

```
#utils/templatetags/utility_tags.py
# -*- coding: UTF-8 -*-
from django.db import models
from django import template
register = template.Library()

### TAGS ###

@register.tag
def get_objects(parser, token):
    """
        Gets a queryset of objects of the model specified by app and
    model names
    Usage:
        {% get_objects [<manager>.]<method> from <app_name>.<model_name> [limit <amount>]
    Example:
        {% get_objects latest_published from people.Person limit 3 as people %}
        {% get_objects site_objects.all from news.Article limit 3 as articles %}
        {% get_objects site_objects.all from news.Article as articles %}
    """
    amount = None
    try:
        tag_name, manager_method, str_from, appmodel, str_limit,\
        amount, str_as, var_name = token.split_contents()
    except ValueError:
        try:
            tag_name, manager_method, str_from, appmodel, str_as,\
            var_name = token.split_contents()
        except ValueError:
            raise template.TemplateSyntaxError, \
                "get_objects tag requires a following syntax: "
                "{% get_objects [<manager>.]<method> from "\
                "<app_ name>.<model_name>"\
                " [limit <amount>] as <var_name> %}"
    try:
        app_name, model_name = appmodel.split(".")
    except ValueError:
        raise template.TemplateSyntaxError, \
            "get_objects tag requires application name and "\
            "model name separated by a dot"
    model = models.get_model(app_name, model_name)
    return ObjectsNode(model, manager_method, amount, var_name)
```

1. 如下，接着在同样的文件中创建节点类：

```
class ObjectsNode(template.Node):
    def __init__(self, model, manager_method, amount, var_name):
        self.model = model
        self.manager_method = manager_method
        self.amount = amount
        self.var_name = var_name

    def render(self, context):
        if "." in self.manager_method:
            manager, method = self.manager_method.split(".")
        else:
            manager = "_default_manager"
            method = self.manager_method

        qs = getattr( qs = getattr(
            getattr(self.model, manager),
            method,
            self.model._default_manager.none,
        )()
        if self.amount:
            amount = template.resolve_variable(self.amount,
                context)
            context[self.var_name] = qs[:amount]
        else:
            context[self.var_name] = qs
        return ''
```

工作原理

The `{% get_objects %}` template tag loads a `QuerySet` defined by the `manager` method from a specified app and model, limits the result to the specified amount, and saves the result to a context variable.

This is the simplest example of how to use the template tag that we have just created. It will load five news articles in any template using the following snippet:

```
{% load utility_tags %}
{% get_objects all from news.Article limit 5 as latest_articles %}
{% for article in latest_articles %}
    <a href="{{ article.get_url_path }}">{{ article.title }}</a>
{% endfor %}
```

This is using the all method of the default `objects` manager of the `Article` model, and will sort the articles by the `ordering` attribute defined in the `Meta` class.

A more advanced example would be required to create a custom manager with a custom method to query objects from the database. A manager is an interface that provides database query operations to models. Each model has at least one manager called `objects` by default. As an example, let's create the `Artist` model, which has a `draft` or `published` status, and a new manager, `custom_manager`, which allows you to select random published artists:
更高级的一个例子是，按要求使用定制方法查询数据库对象以创建一个定制的管理器。管理

器是一个提供对模型进行数据库查询操作的接口。例如，我们创建 `Artist` 模型，它含有一个 `draft` 或者 `published` 状态，新的管理器 `custom_manager` 允许你随机地选择一位艺术工作者：

```
#artists/models.py
# -*- coding: UTF-8 -*-
from django.db import models
from django.utils.translation import ugettext_lazy as _

STATUS_CHOICES = (
    ('draft', _("Draft"),
    ('published', _("Published"),
)
class ArtistManager(models.Manager):
    def random_published(self):
        return self.filter(status="published").order_by('?')

class Artist(models.Model):
    # ...
    status = models.CharField(_("Status"), max_length=20, choices=STATUS_CHOICES)
    custom_manager =  ArtistManager()
```

To load a random published artist, you add the following snippet to any template:

```
{% load utility_tags %}
{% get_objects custom_manager.random_published from artists.Artist limit 1 as random_arti
{% for artist in random_artists %}
    {{ artist.first_name }} {{ artist.last_name }}
{% endfor %}
```

Let's look at the code of the template tag. In the parsing function, there is one of two formats expected: with the limit and without it. The string is parsed, the model is recognized, and then the components of the template tag are passed to the `ObjectNode` class.

让我来看看那模板标签的代码。在解析函数中，有两种格式的一种是需要的：有限制的，或者干脆就没有。只要字符串被解析出来，模型就会识别它，然后模板标签的组件被传递到 `ObjectNode` 类。

In the render method of the node class, we check the manager's name and its method's name. If this is not defined, `_default_manager` will be used, which is, in most cases, the same as objects. After that, we call the manager method and fall back to empty QuerySet if the method doesn't exist. If the limit is defined, we resolve the value of it and limit the `QuerySet`. Lastly, we save the `QuerySet` to the context variable.

参阅

The Creating a template tag to include a template if it exists recipe
The Creating a template tag to parse content as a template recipe
The Creating a template tag to modify request query parameters recipe

创建模板标签并将内容解析为模板

In this recipe, we will create a template tag named {% parse %}, which allows you to put template snippets into the database. This is valuable when you want to provide different content for authenticated and non-authenticated users, when you want to include a personalized salutation, or when you don't want to hardcode media paths in the database.

准备开始

No surprise, we will start with the `utils` app that should be installed and ready for custom template tags.

你不必如此讶异，我们依旧从安装好的`utils`应用开始，并为定制模板标签做好准备。

工作原理

Template tags consist of two things: the function parsing the arguments of the template tag and the node class that is responsible for the logic of the template tag as well as for the output. Perform the following steps:

1. First, let's create the function parsing the template-tag arguments, as follows:

```
#utils/templatetags/utility_tags.py
# -*- coding: UTF-8 -*-
from django import template
register = template.Library()

### TAGS ###

@register.tag
def parse(parser, token):
    """
        Parses the value as a template and prints it or saves to a
    variable
    Usage:
        {% parse <template_value> [as <variable>] %}
    Examples:
    {% parse object.description %}
        {% parse header as header %}
        {% parse "{{ MEDIA_URL }}js/" as js_url %}
    """
    bits = token.split_contents()
    tag_name = bits.pop(0)
    try:
        template_value = bits.pop(0)
        var_name = None
        if len(bits) == 2:
            bits.pop(0)  # remove the word "as"
            var_name = bits.pop(0)
    except ValueError:
        raise template.TemplateSyntaxError, \
            "parse tag requires a following syntax: "\
            "{% parse <template_value> [as <variable>] %}"

    return ParseNode(template_value, var_name)
```

1. Then, we create the node class in the same file, as follows:

```
class ParseNode(template.Node):
    def __init__(self, template_value, var_name):
        self.template_value = template_value
        self.var_name = var_name

    def render(self, context):
        template_value = template.resolve_variable(
            self.template_value, context)
        t = template.Template(template_value)
        context_vars = {}
        for d in list(context):
            for var, val in d.items():
                context_vars[var] = val
                result = t.render(template.RequestContext(
            context['request'], context_vars))
        if self.var_name:
            context[self.var_name] = result
            return ""
        return result
```

工作原理

The `{% parse %}` template tag allows you to parse a value as a template and to render it immediately or to save it as a context variable.

If we have an object with a description field, which can contain template variables or logic, then we can parse it and render it using the following code:

```
{% load utility_tags %}
{% parse object.description %}
It is also possible to define a value to parse using a quoted string like this:

{% load utility_tags %}
{% parse "{{ STATIC_URL }}site/img/" as img_path %}
<img src="{{ img_path }}someimage.png" alt="" />
```

Let's have a look at the code of the template tag. The parsing function checks the arguments of the template tag bit by bit. At first, we expect the name parse, then the template value, then optionally the word as, and lastly the context variable name. The template value and the variable name are passed to the ParseNode class. The render method of that class at first resolves the value of the template variable and creates a template object out of it. Then, it renders the template with all the context variables. If the variable name is defined, the result is saved to it; otherwise, the result is shown immediately.

参阅

The Creating a template tag to include a template if it exists recipe
The Creating a template tag to load a QuerySet in a template recipe
The Creating a template tag to modify request query parameters recipe

创建一个模板标签以修改**request**查询参数

Django has a convenient and flexible system to create canonical, clean URLs just by adding regular expression rules in the URL configuration files. But there is a lack of built-in mechanisms to manage query parameters. Views such as search or filterable object lists need to accept query parameters to drill down through filtered results using another parameter or to go to another page. In this recipe, we will create a template tag named `{% append_to_query %}` , which lets you add, change, or remove parameters of the current query.

准备开始

Once again, we start with the utils app that should be set in INSTALLED_APPS and should contain the templatetags package.

Also, make sure that you have the request context processor set for the TEMPLATE_CONTEXT_PROCESSORS setting, as follows:

```
#settings.py
TEMPLATE_CONTEXT_PROCESSORS = (
    "django.contrib.auth.context_processors.auth",
    "django.core.context_processors.debug",
    "django.core.context_processors.i18n",
    "django.core.context_processors.media",
    "django.core.context_processors.static",
    "django.core.context_processors.tz",
    "django.contrib.messages.context_processors.messages",
    "django.core.context_processors.request",
)
```

如何做

For this template tag, we will be using the simple_tag decorator that parses the components and requires you to define just the rendering function, as follows:

```
#utils/templatetags/utility_tags.py
# -*- coding: UTF-8 -*-
import urllib
from django import template
from django.utils.encoding import force_str
register = template.Library()

### TAGS ###

@register.simple_tag(takes_context=True)
def append_to_query(context, **kwargs):
    """ Renders a link with modified current query parameters """
    query_params = context['request'].GET.copy()
    for key, value in kwargs.items():
        query_params[key] = value
    query_string = u""
    if len(query_params):
        query_string += u"?%s" % urllib.urlencode([
            (key, force_str(value)) for (key, value) in
            query_params. iteritems() if value
            “
```

For this template tag, we will be using the simple_tag decorator that parses the components and requires you to define just the rendering function, as follows:

```
#utils/templatetags/utility_tags.py
# -*- coding: UTF-8 -*-
import urllib
from django import template
from django.utils.encoding import force_str
register = template.Library()

### TAGS ###

@register.simple_tag(takes_context=True)
def append_to_query(context, **kwargs):
    """ Renders a link with modified current query parameters """
    query_params = context['request'].GET.copy()
    for key, value in kwargs.items():
        query_params[key] = value
    query_string = u""
    if len(query_params):
        query_string += u"?%s" % urllib.urlencode([
            (key, force_str(value)) for (key, value) in
            query_params. iteritems() if value
        ]).replace('&', '&amp;')
    return query_string
```

工作原理

The `{% append_to_query %}` template tag reads the current query parameters from the `request.GET` dictionary-like QueryDict object to a new dictionary named query_params, and loops through the keyword parameters passed to the template tag updating the values. Then, the new query string is formed, all spaces and special characters are URL-encoded, and ampersands connecting query parameters are escaped. This new query string is returned to the template.

To read more about `QueryDict` objects, refer to the official Django documentation:

https://docs.djangoproject.com/en/1.6/ref/request-response/#querydict-objects

Let's have a look at an example of how the `{% append_to_query %}` template tag can be used. If the current URL is `http://127.0.0.1:8000/artists/?category=fine-art&page=1`, we can use the following template tag to render a link that goes to the next page:

```
{% load utility_tags %}
<a href="{% append_to_query page=2 %}">2</a>
```

The following is the output rendered, using the preceding template tag:

```
<a href="?category=fine-art&amp;page=2">2</a>
```

Or we can use the following template tag to render a link that resets pagination and goes to another category:

```
{% load utility_tags i18n %}
<a href="{% append_to_query category="sculpture" page="" %}">{% trans "Sculpture" %}</a>
```

The following is the output rendered, using the preceding template tag:

```
<a href="?category=sculpture">Sculpture</a>
```

参阅

The Filtering object lists recipe in Chapter 3, Forms and Views
The Creating a template tag to include a template if it exists recipe
The Creating a template tag to load a QuerySet in a template recipe
The Creating a template tag to parse content as a template recipe

第六章— 模型管理

本章我们覆盖以下议题：

Customizing columns in the change list page
Creating admin actions
Developing change list filters
Exchanging administration settings for external apps
Inserting a map into a change form

引言

The Django framework comes with a built-in administration system for your models. With very little effort, you can set up filterable, searchable, and sortable lists for browsing your models, and configure forms for adding and editing data. In this chapter, we will go through advanced techniques to customize administration by developing some practical cases.

Django框架为你的模型提供了一个内建的管理系统。只需少许努力你就可以为模型浏览配置一个可过滤的，可搜索的，可排序的列表，以及配置可以添加和编辑数据的表单。在这一章，我们会通过编写一个实际的例子来彻底了解定制管理所需的高级技术。

定制切换列表页面的中的列

Change list views in the default Django administration system let you have an overview of all instances of specific models. By default, the model admin property, `list_display`, controls which fields to show in different columns. But additionally, you can have custom functions set there that return data from relations or display custom HTML. In this recipe, we will create a special function for the `list_display` property that shows an image in one of the columns of the list view. As a bonus, we will make one field editable directly in the list view by adding the `list_editable` setting.

改变默认的Django管理系统中的列表试图能够让你拥有一个特定模型的全部实例的概览。默认，模型admin的特性`list_display`控制着在不同的列中哪一个字段会被显示。此外，你可以定制函数

开始前的准备

To start with, make sure that django.contrib.admin is in `INSTALLED_APPS` in the settings, and AdminSite is hooked into the URL configuration. Then, create a new app named products and put it under `INSTALLED_APPS`. This app will have the Product and ProductPhoto models, where one product might have multiple photos. For this example, we will also be using UrlMixin, which was defined in the Creating a model mixin with URL-related methods recipe in Chapter 2, Database Structure.

要准备开始的话，请确保`django.contrib.admin`在设置文件的`INSTALLED_APPS`中，而且Admin站点已经挂在URL配置中了。然后创建一个新的名称为products的应用并把它放到`INSTALLED_APPS`中去。该应用拥有模型`Prodcut`和`ProdcutPhoto`，这里一个procut可能拥有多张照片。就我们的这个例子而言，我们也会用到UrlMixin，

Let's create the Product and ProductPhoto models in the models.py file as follows:

如下，我们在models.py文件中模型Prodcut和ProdcutPhoto：

```
#products/models.py
# -*- coding: UTF-8 -*-
import os
from django.db import models
from django.utils.timezone import now as timezone_now
from django.utils.translation import ugettext_lazy as _
from django.core.urlresolvers import reverse
from django.core.urlresolvers import NoReverseMatch
from utils.models import UrlMixin

def upload_to(instance, filename):
    now = timezone_now()
    filename_base, filename_ext = os.path.splitext(filename)
    return "products/%s/%s%s" % (
        instance.product.slug,
        now.strftime("%Y%m%d%H%M%S"),
        filename_ext.lower(),
    )

class Product(UrlMixin):
    title = models.CharField(_("title"), max_length=200)
    slug = models.SlugField(_("slug"), max_length=200)
    description = models.TextField(_("description"), blank=True)
    price = models.DecimalField(_(u"price (€)"), max_digits=8,
        decimal_places=2, blank=True, null=True)

    class Meta:
        verbose_name = _("Product")
        verbose_name_plural = _("Products")

    def __unicode__(self):
        return self.title

    def get_url_path(self):
        try:
            return reverse("product_detail", kwargs={
                "slug": self.slug})
        except NoReverseMatch:
            return ""

class ProductPhoto(models.Model):
    product = models.ForeignKey(Product)
    photo = models.ImageField(_("photo"), upload_to=upload_to)

    class Meta:
        verbose_name = _("Photo")
        verbose_name_plural = _("Photos")

    def __unicode__(self):
        return self.photo.name
```

具体做法

We will create a simple administration for the Product model that will have instances of the ProductPhoto model attached to the product as inlines.

In the `list_display` property, we will define the get_photo method name of the model admin that will be used to show the first photo from the many-to-one relationship.

Let's create an admin.py file with the following content:

```
#products/admin.py
# -*- coding: UTF-8 -*-
from django.db import models
from django.contrib import admin
from django.utils.translation import ugettext_lazy as _
from django.http import HttpResponse

from products.models import Product, ProductPhoto

class ProductPhotoInline(admin.StackedInline):
    model = ProductPhoto
    extra = 0

class ProductAdmin(admin.ModelAdmin):
    list_display = ["title", "get_photo", "price"]
    list_editable = ["price"]

    fieldsets = (
        (_("Product"), {
            "fields": ("title", "slug", "description", "price"),
        }),
    )
    prepopulated_fields = {"slug": ("title",)}
    inlines = [ProductPhotoInline]

    def get_photo(self, obj):
        project_photos = obj.productphoto_set.all()[:1]
        if project_photos.count() > 0:
            return u"""<a href="%(product_url)s" target="_blank">
                <img src="%(photo_url)s" alt="" width="100" />
            </a>""" % {
                "product_url": obj.get_url_path(),
                "photo_url":  project_photos[0].photo.url,
            }
        return u""

  get_photo.short_description = _("First photo")
  get_photo.allow_tags = True

admin.site.register(Product, ProductAdmin)
```

工作原理

If you look at the product administration list in the browser, it will look like this:

图片：略

Besides the normal field names, the `list_display` property accepts a function or another callable, the name of an attribute of the admin model, or the name of the attribute of the model. Each callable will be passed a model instance as the first argument. So, in our example, we have the `get_photo` method of the model admin that retrieves Product as obj. The method tries to get the first ProductPhoto from the `many-to-one` relation and, if it exists, it returns HTML with the tag linked to the detail page of Product.

The `short_description` property of the callable defines the title shown for the column. The allow_tags property tells the administration not to escape the HTML values.

In addition, the price field is made editable by the `list_editable` setting and there is a save button at the bottom to save the whole list of products.

参阅

- The Creating a model mixin with URL-related methods recipe in Chapter 2, Database Structure
- The Creating admin actions recipe
- The Developing change list filters recipe

添加Admin的行为

The Django administration system provides actions that we can execute for selected items in the list. There is one action given by default, and it is used to delete selected instances. In this recipe, we will create an additional action for the list of the Product model that allows administrators to export selected products to Excel spreadsheets.

开始前的准备

我们就从之前方法中所创建的应用`products`开始。

具体做法

Admin actions are functions that take three arguments: the current ModelAdmin value, the current HttpRequest value, and the QuerySet value containing the selected items. Perform the following steps:

Let's create an export_xls function in the admin.py file of the products app, as follows:

```
#products/admin.py
# -*- coding: UTF-8 -*-
import xlwt
# ... other imports ...

def export_xls(modeladmin, request, queryset):
    response = HttpResponse(mimetype="application/ms-excel")
    response["Content-Disposition"] = "attachment; "\
        "filename=products.xls"
    wb = xlwt.Workbook(encoding="utf-8")
    ws = wb.add_sheet("Products")

    row_num = 0
  ### Print Title Row ###
  columns = [
        # column name, column width
        (u"ID", 2000),
        (u"Title", 6000),
        (u"Description", 8000),
        (u"Price (€)", 3000),
    ]

    header_style = xlwt.XFStyle()
    header_style.font.bold = True

    for col_num in xrange(len(columns)):
        ws.write(row_num, col_num, columns[col_num][0],
            header_style)
        # set column width
        ws.col(col_num).width = columns[col_num][1]

    ### Print Content ###

    text_style = xlwt.XFStyle()
    text_style.alignment.wrap = 1

    price_style = xlwt.XFStyle()
    price_style.num_format_str = "0.00"

    styles = [text_style, text_style, text_style, price_style]
    for obj in queryset.order_by("pk"):
        row_num += 1
        row = [
            obj.pk,
            obj.title,
            obj.description,
            obj.price,
        ]
        for col_num in xrange(len(row)):
            ws.write(row_num, col_num, row[col_num],
                styles[col_num])

    wb.save(response)
    return response

export_xls.short_description = u"Export XLS”
```

1. Then, add the actions setting to ProductAdmin, as follows:

```
class ProductAdmin(admin.ModelAdmin):
    # ...
    actions = [export_xls]
```

工作原理

If you look at the product administration list page in the browser, you will see a new action called Export XLS along with the default action Delete selected Products:

图片：略

By default, admin actions do something with QuerySet and redirect the administrator back to the change list page. However, for some more complex actions like this, HttpResponse can be returned. The export_xls function returns HttpResponse with the MIME type of Excel spreadsheet. Using the Content-Disposition header, we set the response to be downloadable with the file named products.xls.

Then, we use the xlwt Python module to create the Excel file.

At first, the workbook with UTF-8 encoding is created. Then, we add a sheet named Products to it. We will be using the write method of the sheet to set the content and style for each cell and the col method to retrieve the column and set the width to it.

To have an overview of all columns in the sheet, we create a list of tuples with column names and widths. Excel uses some magical units for the widths of the columns. They are 1/256 of the width of the zero character for the default font. Next, we define the header style to be bold.

As we have the columns defined, we loop through them and fill the first row with the column names, also assigning the bold style to them.

Then, we create a style for normal cells and for the prices. The text in normal cells will be wrapped in multiple lines. Prices will have a special number style with two points after the decimal point.

Lastly, we go through the QuerySet of the selected products ordered by ID and print the specified fields into corresponding cells, also applying specific styles.

The workbook is saved to the file-like HttpResponse object and the resulting Excel sheet looks like this:

表格：略

参阅

- Chapter 9, Data Import and Export
- The Customizing columns in the change list page recipe
- The Developing change list filters recipe

开发

本章我们会学习以下内容：

``` Creating templates for Django CMS
为Django CMS创建模板

Structuring the page menu
组织页面菜单

Converting an app to a CMS app 把一个应用转换到CMS应用

Attaching your own navigation 加入用户自己的导航

Writing your own CMS plugin
编写用户自己的CMS插件

Adding new fields to the CMS page
对CMS页面添加新字段
```

本章-我们会学习以下内容：

- 创建分层类别
- 用django-mptt-admin生成分类admin接口
- 用django-mptt-tree创建分类admin接口
- 在模板中传递分类
- 在表单中使用一个单选字段选择一个分类
- 在表单中使用一个复选框列表选择多个分类

本章，我们会学习以下技巧：

- 从一个本地CSV文件导入数据
- 从一个本地Excel文件导入数据
- 从一个外部JSON文件导入数据
- 从一个外部XML文件导入数据
- 生成一个可过滤的RSS订阅
- 使用Tastypie为第三方提供数据

在本章，我们将学习以下技巧：

- 使用Django Shell
- The monkey patching slugification function
- The monkey patching model administration
- Toggling Debug Toolbar
- Using ThreadLocalMiddleware
- Caching the method value
- 通过电子邮件获取错误报告详情
- 使用 mod_wsgi 在Apache上部署项目
- 创建并使用Farbic部署脚本

引言

In this chapter, we will go through several other important bits and pieces that will help you understand and utilize Django even better. You will get an overview of how to use the Django shell to experiment with the code before writing it into files. You will be introduced to monkey patching, also known as guerrilla patching, which is a powerful feature of dynamical languages such as Python, but should be used with moderation. You will learn how to debug your code and check its performance. Lastly, you will be taught how to deploy your Django project on a dedicated server.

使用Django shell

With the virtual environment activated and your project directory selected as the current directory, enter this command into your command-line tool:

```
(myproject_env)$ python manage shell
```

By executing the preceding command, you get into an interactive Python shell configured for your Django project, where you can play around with the code and inspect classes, try out methods, or execute scripts on the fly. In this recipe, we will go through the most important functions you need to know to work with the Django shell.

准备开始

Optionally, you can install IPython or bpython using one of the following commands, which will highlight the syntax for the output of your Django shell:

```
(myproject_env)$ pip install ipython
(myproject_env)$ pip install bpython
```

具体实现过程

你可以通过下面的这些说明学习到Django shell的使用基础：

1. 输入下面的命令以运行Django shell：

```
(myproject_env)$ python manage shell
```

The prompt will change to In [1]: or >>> depending on whether you use IPython or not. Now you can import classes, functions, or variables and play around with them. For example, to see the version of an installed module, you can import the module and then try to read its **version**, VERSION, or version variables:

```
>>> import MySQLdb
>>> MySQLdb.__version__
'1.2.3'
```

To get a comprehensive description of a module, class, function, method, keyword, or documentation topic, use the help function. You can either pass a string with the path to a specific entity, or the entity itself, as follows:

```
>>> help('django.forms')
```

This will open the help page for the django.forms module. Use the arrow keys to scroll the page up and down. Press Q to get back to the shell. This is an example of passing an entity to the help function. This will open a help page for the ModelForm class:

```
>>> from django.forms import ModelForm
>>> help(ModelForm)
```

To quickly see what fields and values are available for a model instance, use the **dict** attribute. Also use the pprint function to get dictionaries printed in a more readable format (not just one long line):

```
>>> from pprint import pprint
>>> from django.contrib.contenttypes.models import ContentType
>>> pprint(ContentType.objects.all()[0].__dict__)
{'_state': <django.db.models.base.ModelState object at 0x10756d250>,
 'app_label': u'bulletin_board',
 'id': 11,
 'model': u'bulletin',
 'name': u'Bulletin'}
```

Note that by using this method, we don't get many-to-many relationships. But this might be enough for a quick overview of the fields and values.

To get all the available properties and methods of an object, you can use the dir function, as follows:

```
>>> dir(ContentType())
['DoesNotExist', 'MultipleObjectsReturned', '__class__', '__delattr__', '__dict__', '__do
'sion_set', 'pk', 'prepare_database_save', 'save', 'save_base', 'serializable_value', 'un
```

The Django shell is useful for experimenting with QuerySets or regular expressions before putting them into your model methods, views, or management commands. For example, to check the e-mail-validation regular expression, you can type this into the Django shell:

```
>>> import re
>>> email_pattern = re.compile(r'[^@]+@[^@]+\.[^@]+')
>>> email_pattern.match('aidas@bendoraitis.lt')
<_sre.SRE_Match object at 0x1075681d0>
```

To exit the Django shell, press Ctrl + D or type the following command:

> > > exit()

工作原理

The difference between a normal Python shell and the Django shell is that when you run the Django shell, manage.py sets the DJANGO_SETTINGS_MODULE environment variable to the project's settings path, and then Django gets set up. So, all database queries, templates, URL configuration, or anything else are handled within the context of your project.

参见

The The monkey patching slugification function recipe The The monkey patching model administration recipe

The monkey patching slugification function

Monkey patching or guerrilla patching is a piece of code that extends or modifies another piece of code at runtime. It is not recommended to use monkey patching often, but sometimes it is the only possible way to fix a bug in third-party modules without creating a separate branch of the module, or to prepare unit tests for testing without using complex database manipulations. In this recipe, you will learn how to exchange the default slugify

function with the third-party awesome-slugify module, which handles German, Greek, and Russian words smarter and also allows custom slugification for other languages. The slugify function is used to create a URL-friendly version of the object's title or the uploaded filename: it strips the leading and trailing whitespace, converts the text to lowercase, removes nonword characters, and converts spaces to hyphens.